

LA PRIMERA REVISTA DE PROGRAMACION EN CASTELLANO
SÓLO PROGRAMADORES
por desarrolladores ESCRITA para desarrolladores
Revista especializada para usuarios de PC
AÑO 3. Nº 28
1250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $
PORTUGAL 1500$
ANALIZAMOS EL ESTADO DE LA INTELIGENCIA ARTIFICIAL
VISUAL C++ PROGRAMACIÓN DEL GDI
COMUNICACIONES EN WINDOWS
Y además:
Chequeo a la red Internet
Protocolos de comunicaciones
Grandes sistemas
CURSOS PRÁCTICOS DE:
Programación de demos
Tecnología WEB
Visual Basic 4.0
Programación básica
Y mucho más...
TOWER COMMUNICATIONS, S.R.L
GRATIS CD-ROM CON:
FRONTPAGE 97
MICROSOFT ACTIVEX SDK
PROGRAMACIÓN EN INTERNET...
ESPECIAL HERRAMIENTAS R.A.D.
DESARROLLO RÁPIDO DE APLICACIONES CON OPTIMA C++
1996 Movie Sales Per Quarter
8 413042 303299
00028



Asistencia Técnica y Formación

Soporte Técnico Borland®

Estimado usuario.

El continuo avance en las tecnologías informáticas, nos ofrecen cada día unas herramientas de trabajo más sofisticadas, así como la necesidad de actualizar los conocimientos.

El Soporte Técnico Borland, está destinado para todos aquellos que trabajan con la informática de forma profesional. Nuestro labor es proporcionarles ayuda cualificada para obtener el mayor rendimiento de las herramientas, agilizar su trabajo evitándole perder su tiempo en encontrar soluciones, y procurarle el tránsito más rápido y fácil a éstas nuevas tecnologías.

Con este fin, queremos darle a conocer nuestros servicios, que abarcan asistencia técnica, y formación. Además, encontrará que inscribirse al Soporte Técnico, le supone otras muchas ventajas.

Para todos nuestros socios Cuenta E-mail y Kit de conexión
Internet Gratis

Si desea recibir información detallada, y Agenda de nuestros Seminarios, Cursos de programación, y presentaciones de productos, póngase en contacto con nosotros en:

http: //www.databasedm.es
(también en Infovía)
Tel.: 902 10 20 77
Fax: 902 10 20 66
E-mail: soporte@databasedm.es





DICIEMBRE 1996. Número 28
SÓLO PROGRAMADORES es una publicación de Tower Communications

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Coordinador Técnico
Eduardo Toribio Pazos
Edición
Miguel Cabezuelo
Colaboradores
F. de la Villa, Fernando J. Echevarrieta, Pedro Antón. J. M. Martín, L. Martín, J. M. Peco, José C. Remiro, Jorge del Río, Enrique De Alarcón, David Aparicio, M. Jesús Recio, Carlos Arias

Jefe de Diseño
Fernando García Santamaría
Maquetación
Clara Francés
Tratamiento de Imagen
María Arce Giménez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Publicidad
Papín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Isabel Bojo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Filmación
Filma Dos S.L.
Impresión
G. Reunidas
Distribución
SGEL

Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
Director de Distribución
Enrique Cabezas García
Asesor Editorial
Mario de Luis
Redacción, Publicidad y Administración
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792



EDITORIAL

LA FORMA Y EL FONDO

Una revista, para sobrevivir en el mercado, debe vender, para lo cual debe ser atractiva para el lector. Hoy por hoy, acercarse a un quiosco cualquiera es hasta peligroso debido a la multitud de revistas que se venden en descomunales cartonajes de afilados picos, con multitud de CDs de diversos colores y difícil extracción. De acuerdo, es importante la forma, pero ¿y el fondo?, ¿qué pasa con lo que de verdad debe ser el verdadero gancho para el lector ?. Esta pequeña reflexión viene a colación de lo siguiente :

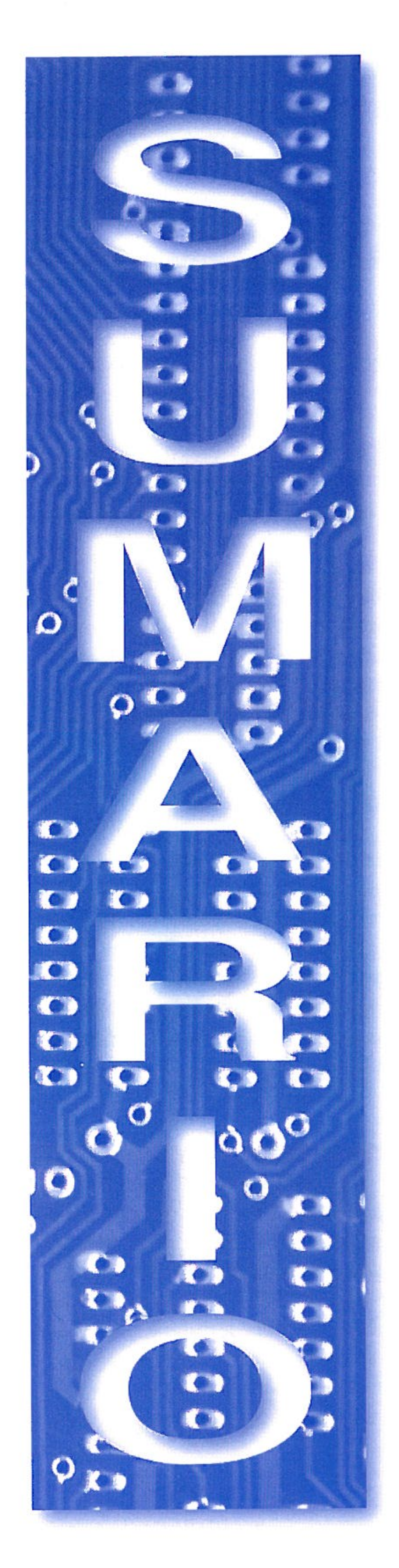
Personalmente, siempre me ha preocupado la dificultad de conseguir el primer puesto de trabajo que tienen los jóvenes españoles en general, y por tanto los jóvenes informáticos en particular (quizá por ser joven e informático). Me preocupa que gente que ha estudiado durante tantos años una carrera técnica de alta dificultad, no tengan en muchas ocasiones un acceso fácil al mercado laboral, ni una preparación adecuada para entrar en este mundillo. Pero eso es otro asunto, nosotros no podemos hacer mucho para solucionar este problema, pero podemos aportar un pequeño granito de arena de diversas maneras.

La primera de ella, lógicamente, es mediante el contenido de la revista. Cualquier lector que ojee un periódico de los llamados de ofertas de empleo comprobará que existen ofertas que demandan ciertos conocimientos que se repiten en la mayoría de los anuncios, por ejemplo: Visual Basic, Visual C++, Grandes sistemas, SQL, etc. Bien, la mayor parte de estos conocimientos necesarios para obtener el anhelado puesto de trabajo intentamos sean difundidos por nuestra revista. Si nosotros comprobamos que surge un nuevo campo, herramienta o lenguaje con demamanda de empleo, intentamos rápidamente su inmediata aparición en la revista.

La segunda es una nueva forma que a partir de este mes vamos a incluir en nuestro CD-ROM, un apartado muy especial que llamaremos "Guía-SP de Informáticos" y que consistirá en una serie de páginas HTML preparadas para navegación off-line de manera cómoda, con los datos de jóvenes y pronto no tan jóvenes programadores que posean una media de preparación buena-alta. **Sólo** será para **Programadores,** independientemente de que éstos sean o provengan de la Facultad o Escuela de Informática, de Teleco, Industriales, FP o de las distintas academias. Eso sí, deberán ser programadores, no ofimáticos. El objetivo es que las distintas empresas que necesiten un programador, ya sea de alto nivel o bajo o multimedia, encuentren a su candidato ideal en el CD de la revista, teniendo la confianza de que las personas allí reflejadas son programadores de verdad.

Para finalizar, algo también importante. Los que formamos la redacción de esta revista deseamos una feliz Navidad a todos los lectores de Sólo Programadores, a todos aquellos que trabajan en esta revista en cualquiera de sus departamentos, y muy especialmente a todos los colaboradores de la revista.

Eduardo Toribio
etori@ergos.es



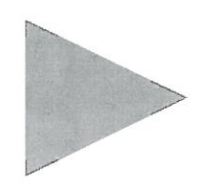





NOTICIAS:

NOVEDADES DEL MERCADO

Espacio destinado a informar al lector de las últimas novedades acontecidas en el mundo de la informática y el comentario de los libros de interés.





TECNOLOGÍA WEB (XIII):

METODOLOGÍA Y TRANSMISIÓN DE ESTADO



En los últimos meses, en esta sección, se ha profundizado es la programación CGI. Se considera ahora la metodología de programación asociada.



CURSO DE PROGRAMACIÓN (XXII):

ARCHIVOS INDEXADOS (I)

La forma en que está organizada la información en un archivo, a veces, no es suficiente para acceder a ella de forma eficaz. Estudiamos el problema.





SISTEMAS ABIERTOS (XXII):

CHEQUEO A INTERNET (III)

Este mes continúa en la sección la miniserie dedicada a los protocolos de Internet presentando el protocolo TCP.

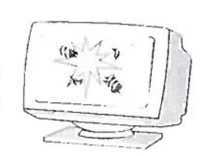




CURSO DE DEMOS (XVIII)

AÑADIENDO GRÁFICOS A LA DEMO

Uno de los formatos gráficos más extendidos es el PCX de Zsoft. En este artículo se añadirá a la librería una rutina de carga en formato PCX adaptada a nuestras necesidades.

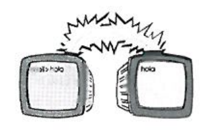




REDES LOCALES (VIII):

PROTOCOLOS DE COMUNICACIONES

De nada sirve tener grandes equipos, tarjetas de red con DMA, si no se tienen protocolos de comunicaciones adecuados y bien configurados en la red.



40

ANALISIS Y ESTUDIOS

INTELIGENCIA ARTIFICIAL

La Inteligencia Artificial es una de las áreas dentro del mundo de la programación que más importancia tendrá en un futuro inmediato.





GRANDES SISTEMAS (XVIII):

UTILIDADES MVS (III)

El presente artículo cierra esta miniserie dedicada a las utilidades del MVS. Algunas quedan en el tintero, pero comentadas quedan las más frecuentes.





HERRAMIENTAS DE DESARROLLO:

OPTIMA++ 1.5

La calidad de las aplicaciones orientadas al RAD está mejorando notablemente. Optima++ es, sin duda, uno de los máximos exponentes de esta filosofía de creación de aplicaciones.

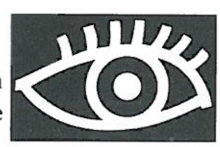




VISUAL C++ 4.0 (XI):

PROGRAMACIÓN DEL GDI (II)

La independencia de las operaciones gráficas en Windows, debido al GDI, tienen su máximo exponente en el tratamiento de los bitmaps.

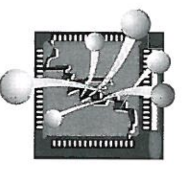




CURSO DE VISUAL BASIC 4.0 (XIX):

PROCEDIMIENTOS

En este artículo se analiza el funcionamiento de los distintos tipos de procedimientos soportados por Visual Basic, así como sus diferentes variantes y particularidades.

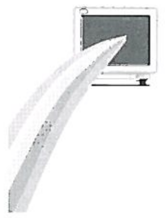




COMUNICACIÒN EN WINDOWS (I):

LOS COMPONENTES TAPI

Durante esta serie de artículos se va a estudiar la interfaz de programación telefónica de Microsoft. Comienza la serie con una introducción.



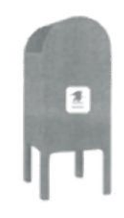


CONTENIDO DEL CD-ROM:

HERRAMIENTAS DE PROGRAMACIÓN

En el CD-ROM de Diciembre las demos de la Euskal Party, la Guía-SP de informáticos y una colección de herramientas y tutoriales de programación.





CORREO DEL LECTOR:

CONSULTAS DE LOS LECTORES

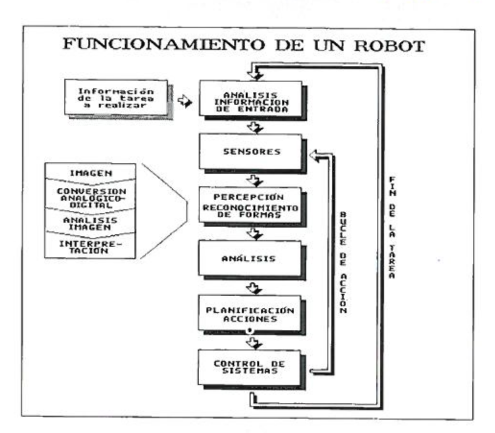
Espacio dedicado a resolver los problemas surgidos a los lectores en diversos aspectos de la programación.

OPTIMA ++ 1.5

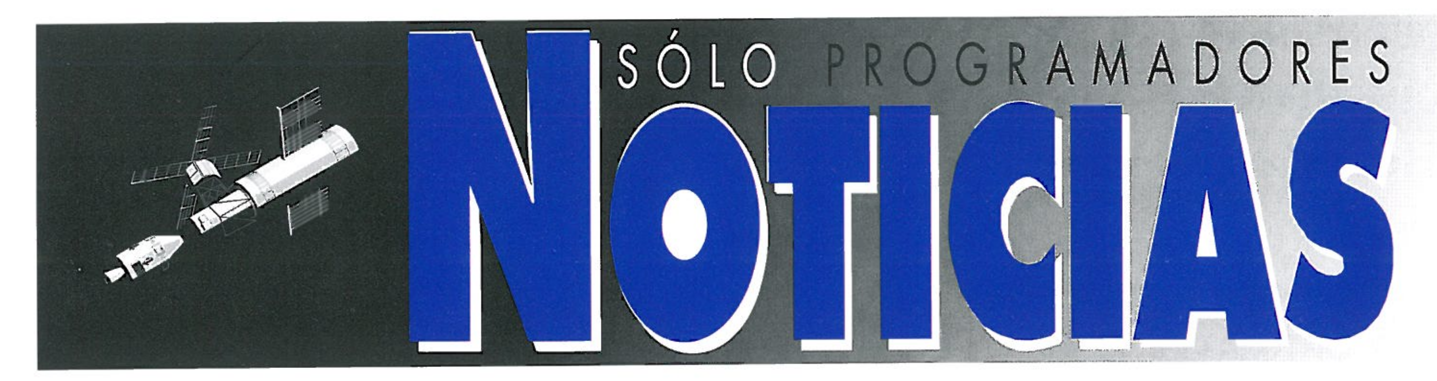


Optima++ se perfila como uno de los mejores entornos para el desarrollo rápido de aplicaciones C++ para Windows. Dispone de un entorno sencillo y poco recargado, pero a la vez muy potente. Los creadores de Optima++ han comprendido realmente lo que significa el acrónimo RAD y han sabido reunir lo mejor de otras herramientas en un mismo producto.

INTELIGENCIA ARTIFICIAL (Pág. X)



Próximamente comenzará en Sólo Programadores, una nueva serie dedicada a la inteligencia artificial en la cual estudiaremos alguno de los lenguajes de programación más habituales. Para abrir boca, este mes, un análisis del pasado y presente de esta apasionante materia.



OBJECTIVE COBOL
Programación basada en objetos

Nigsun Internacional ha presentado recientemente Objective Cobol, la nueva programación basada en objetos y eventos para el lenguaje Cobol. Esta nueva herramienta permite a los programadores desarrollar rápidamente aplicaciones RAD bajo plataformas Windows 3.x, 95 y NT. Entre las principales características de este paquete destacan las siguientes: sintaxis Cobol, módulo Runtime libre de royalties, motor de bases de datos relacionales, entorno integrado de desarrollo, generador de informes y controles de gráficos, hoja de cálculo, etc... Además, Objective Cobol incorpora más de 60 objetos para el diseño y más de 60 propiedades y eventos aplicables a dichos objetos.

Para más información:
NIGSUN INTERNACIONAL
Tel: (91) 667 02 50
Fax: (91) 667 02 50

PARTS FOR JAVA
Herramientas de desarrollo visual para Java

ParcPlace-Digitalk ha ampliado su familia Parts de herramientas de desarrollo visual incluyendo Parts for Java. Esta nueva herramienta facilita la cración de applets y aplicaciones en Java proporcionando un completo entorno de desarrollo Java visual, la posibilidad de crear, explorar y comprender código Java y un extenso sistema para añadir componentes Java a la paleta de desarrollo. Así mismo, Parts for Java incluye un browser para visualizar y editar código y clases jerárquicas, un conjunto de herramientas abierto para seleccionar compiladores y otras utilidades, y un modo de trabajo para contruir apllets y aplicaciones desde otros componentes.
http://www.parcplace.com/

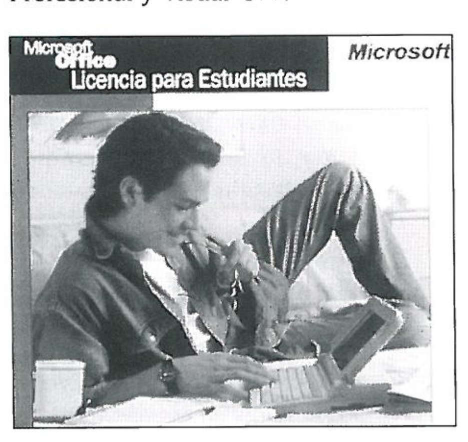
LICENCIA PARA ESTUDIANTES
Microsoft abarata el precio del software

Microsoft Ibérica ha lanzado su Licencia para Estudiantes, un programa de licencias de software que hace posible el acceso de los estudiantes a las últimas novedades de la multinacional a un precio realmente asequible para ellos, con un beneficio de más del ochenta por ciento de descuento en los precios de estos productos. Los beneficiarios de estas licencias serán todos los estudiantes universitarios, los de Formación Profesional y aquellos estudiantes de Escuelas de Negocios que estén interesados en adquirir software legal, con todas las ventajas que esto supone. Para ello será necesario que las personas que deseen beneficiarse de esta nueva tarifa presenten los documentos que les acrediten como estudiantes. Este programa incluye las siguientes licencias:

Licencia para Estudiantes de sistemas operativos: Diseñada para que los estudiantes cuenten con los nuevos sistemas operativos completos. Esta licencia contiene Windows 95 y Windows NT WorkStation.

Licencia para Estudiantes de Microsoft Office: Incluye el procesador de textos Microsoft Word, la hoja de cálculo Microsoft Excel, el programa de presentaciones Microsoft PowerPoint y el gestor de bases de datos Microsoft Access.

Licencia para Estudiantes de herramientas de desarrollo: Los estudiantes de ingenierías diversas, de Informática, de Formación Profesional o todos aquellos que estén interesados en la programación podrán adquirir esta modalidad, que incluye Microsoft Visual Basic Profesional y Visual C++.



WORLDTOOLKIT 6
Nueva herramienta de desarrollo de Realidad Virtual

Sense8 ha comenzado a desarrollar la versión 6 de su software de simulación visual y realidad virtual WorldToolKit. WorldToolKit 6 ofrecerá un rango ensanchado de nodos que podrán ser organizados jerárqucamente en la simulación gráfica. Cada nodo representa una parte de la simulación, y mientras el escenario gráfico es construido nodo a nodo, estos son hechos en WorldToolKit mediante el uso de llamadas API de alto nivel. Además, será posible cargar un fichero VRML desde Internet dentro de un escenario gráfico con sólo utilizar una simple llamada de función. Así mismo, WorldToolKit incluye una librería orientada a objetos escrita en C que contiene más de 650 lamadas a funciones de alto nivel para configuración, interactuando con simulaciones de control de tiempo real, que proporciona la funcionalidad necesaria para crear las más complejas aplicaciones.

WorldToolKit está diseñado respetando la filosofía de OpenVR. Esto significa que Esta herramienta es portable entre varias plataformas, soporta una amplia variedad de periféricos de entrada y salida, habilita al programador a incorporar código existente en C como lectura de ficheros y rutinas de dibujo e interactúa con una amplia gama de fuentes de información. Al mismo tiempo, WorldToolKit está optimizado para premitir el máximo uso de las capacidades de cada plataforma y desarrollar los gráficos más rápidos posibles. Este paquete correrá bajo plataformas Silicon Graphics, Sun, Hewlett-Packard, DEC, Intel y PowerPC, entre otras, y saaldrá a la vez una versión especial que proporcionará soporte para renderizado de pilas de gráficos o escenas múltiples.

Para más información:
http://www.sense8.com/

PURIFY FOR WINDOWS NT
Detección de errores en tiempo de ejecución y escapes de memoria

Pure Software ha anunciado recientemente Purify for Windows NT. Purify apunta directamente los errores en tiempo de ejecución y escapes de memoria a través de aplicaciones C y C++, incluyendo controles ActiveX, componentes OLE y DLLs de tercera generación y librerías que actúan como bloques de construcción de aplicaciones Windows de 32 bits. Purify está basado en una nueva tecnología patentada, llamada Object Code Insertion (OCI). Como OCI no requiere acceso al código fuente, Purify puede localizar errores a través de componentes de tercera generación con los que la mayoría de las aplicaciones Windeows fueron desarrolladas. Con Purify se pueden recoger datos en tiempo de ejecución y apuntar directamente al origen exacto del bug. El programa también identifica dónde no es accesible la memoria a través de algún puntero del programa.

Para más información:
http://www.pure.com/

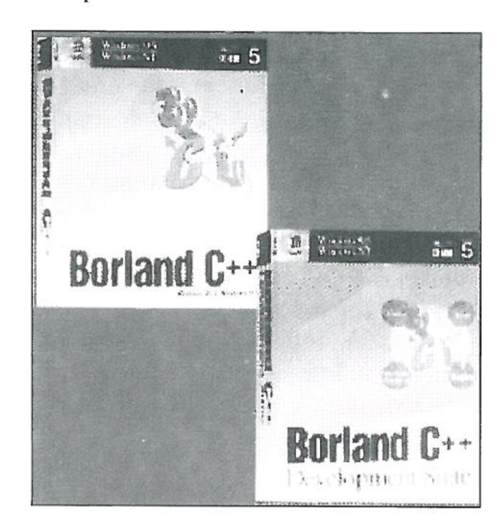
VISUAL DBASE PROFESIONAL CON HERRAMIENTAS INTERNET
Bases de datos interactivas para la WWW

Ya está disponible Visual dBASE Profesional con Herramientas Internet, una nueva versión de Visual dBASE que incluye un conjunto de utilidades que permite a los desarrolladores xBASE crear fácilmente aplicaciones de bases de datos interactivas para la World Wide Web y proporciona a los usuarios el estándar en bases de datos más avanzado junto a un entorno de desarrollo totalmente orientado a objetos y un compilador que permitirá distribuir las aplicaciones libremente. Este nuevo paquete incluye el gestor y el entorno de desarrollo de aplicaciones de bases de datos Visual dBASE 5.5 y el compilador de Visual dBASE. Asimismo, también incluye las Delta Point Web Tools, que son una librería de clases y una colección de utilidades de desarrollo escritas totalmente en código dBASE que proporcionan un modo fácil de publicar tablas de bases de datos para escritorios y servidores en la Web, de forma que los usuarios puedan acceder y hacer consultas en cualquier plataforma con un browser estándar. Estas utilidades no incluyen el código fuente.

La herramienta soporta todos los formatos de bases de datos soportados por el Borland Database Engine incluyendo dBASE, Paradox, ODBC, Sybase, Oracle, Microsoft SQL Server, Informix y Borland Interbase. Un ejemplo de aplicación Web de base de datos escrita con estas herramientas puede ser ejecutada en la dirección *http://webtools.borland.com/wthtml/wt.htm.*

BORLAND C++ 5.01
Nueva actualización del entorno C++ de Borland

Borland ha lanzado la actualización a la versión 5.01 para Windows 95 y Windows NT de su popular entorno de desarrollo Borland C++. Esta nueva versión incluye multitud de herramientas innovadoras para ahorrar tiempo y esfuerzo al programador en el desarrollo de las aplicaciones. Entre estas nuevas mejoras cabe destacar: la fácil migración a Windows 95 realizando un desarrollo simultáneo para Windows NT, 3.1 y DOS desde el mismo entorno, ya sea 32 ó 16 bits; flexibilidad mediante un entorno integrado de desarrollo programable a través del nuevo lenguaje OpenScripting, que da acceso a los mismos objetos que utilizaron los programadores de Borland para desarrollar el IDE; reutilización de código con soporte para VBX y controles OCX, que permite la reutilización de los controles VBX existentes al mismo tiempo que soporta el nuevo estándar OCX (controles ActiveX); posibilidad de compartir un único fuente para las aplicaciones de 16 y 32 bits a través del nuevo OWL 5.0; y programación visual de bases de datos con las Visual Database Tools, un conjunto de herramientas que permitirán acceder a los datos utilizando el API de C, el Object Layer de C++ y la automatización OLE, además de incluir los Borland Database Engine de 16 y 32 bits para aumentar la potencia de las aplicaciones. Además, Borland C++ 5.01 ofrece lo último en tecnología multiplataforma al incluir Java, el lenguaje de programación más robusto y compacto actualmente para el desarrollo de aplicaciones para Internet.



VANGUI INTERFACE BUILDER
Integración de aplicaciones Cobol en Windows

Ryan McFarland, una división de Liant Software, ha lanzado VanGui Interface Builder for Delphi como parte de una suite de herramientas comprensivas conocida como VanGui Product Set. El Interface Builder está diseñado para integrar aplicaciones Cobol con entornos gráficos basados en Windows y tecnologías de acceso a datos para entornos cliente/servidor. Esto habilita a los programadores en Cobol a usar herramientas estándar como Delphi o Visual Basic para desarrollar el entorno Windows para aplicaciones Cobol basadas en Windows o UNIX. Esta nueva herramienta está disponible para aplicaciones Windows o UNIX que corran en plataformas HP/UX, AIX, SCO, Solaris y AT&T SVR4.

Para más información:
http://www.liant.com/rm/

AN INTRODUCTION TO PROGRAMMING JAVA APPLETS
Nueva línea de tutoriales de MindQ

MindQ ha desarrollado un CD-ROM Java multimedia llamado An Introduction to Programming Java Applets. Este CD es el primero de una familia de tutoriales llamada Java Series, destinada a hacer más ameno el aprendizaje de esta tecnología de programación para la WWW al eliminar los libros como método de aprendizaje. Este CD establece un balance entre contenido y presentación para ayudar a los usuarios, tanto expertos como noveles, avanzar en la programación Java de una manera efectiva y acelerada. El interfaz multimedia de este tutorial, con simulación de entornos de desarrollo, links, vídeos y narraciones concisas, facilita la comprensión de los conceptos de Java. La visualización paso a paso ayuda a comprender los objetos, clases, métodos y muchos otros conceptos relacionados con la programación orientada a objetos (OOP). Así mismo, la aparición de código animado explica cómo son escritos los applets Java línea a línea. Esta iniciativa de MindQ supone un paso adelante en los métodos de aprendizaje a través de un sistema más comprensible de entender la programación. Otros títulos disponibles son The Better Way to Learn Java Fast, Java Programming and Core Class Libraries y Hands on Java Programming Using Microsoft Visual J++, también de MindQ. Existen demos de estos CDs en la dirección Web de la compañía para evaluar estos productos.

Para más información:
http://www.mindq.com/

LIB/400
Librería de acceso a bases de datos de la gama AS400

DFL ha lanzado recientemente una nueva librería de programación para Delphi que permitirá el acceso a bases de datos de toda la gama de AS400 de manera similar a las demás bases de datos relacionales soportadas por Delphi. Esta librería, denominada Liub/400, es transparente al Borland Data Engine (BDE), por lo que no se requerirá formación especializada para hacerla funcionar contra los AS400, y al mismo tiempo permite el acceso a bases de datos heterogeneas al mismo tiempo, es decir, utilizar tablas Paradox, InterBase, DBF, etc... junto con las de los AS400. Respecto a su funcionamiento, una parte de la librería, denominada Easycom Database, residirá en el AS400, mientras que la otra formará parte del BDE. Como características principales de Lib/400 destacan la posibilidad de que los programas Delphi utilicen los ficheros lógicos como índices, el acceso a ficheros de consultas (query files), el acceso simultáneo a diferentes AS400, el soporte de ficheros compartidos y bloqueos a nivel de registro, acceso a control de transacciones, instalación y consultas de bases de datos (OPNQRYF, OVRDBF, etc...) y la ventaja de que no se genera ningun tipo de código en el AS400.

Los requerimientos para esta librería para AS400 son: equipo AS400 serie B o superior, OS/400 versión 2.2 o superior y conexión a PC a través de Ethernet, Token Ring, Twinax o redes SDLC, teniendo una sola conexión con cada PC. Respecto a los PCs, los requerimientos son procesador 386 o superior, router APPC (PCS, Client acces o Router NS, Eicon, Netware para SAA, Microsoft Clent SNA o Microsoft SNA Server), Windows 3.x/NT/95 y Delphi para desarrollo en 16 0 32 bits.

El paquete estará disponible en dos versiones: la primera de desarrollo y la otra para clientes de dicho producto. Ambas versiones servidor y cliente disponen de versiones con RPC (Remote Procedure Calls).

VISUAL CAFÉPRO
Nueva herramienta de desarrollo de Symantec Corp

Symantec Corp. ha anunciado recientemente Visual Café Pro, la primera herramienta visual de desarrollo rápido de aplicaciones (RAD) para creación de applets y aplicaciones que conectan con bases de datos relacionales. Tras Visual Café, este nuevo paquete es el siguiente paso de Symantec en la creación de herramientas para desarrollo de applets y aplicaciones.

Entre las nuevas funcionalidades de este producto destaca un nuevo engine de bases de datos, conectividad directa con bases de datos Microsoft Access, Oracle, Microsoft SQL Server y Sybase. Así mismo, incluye una extensión API para Java que soporta complejas bases de datos y un entorno de desarrollo visual de formularios de bases de datos desde bases de datos existentes. Al mismo tiempo, incorpora una serie de asistentes que ayudan a crear automáticamente formularios en todas las tablas de bases de datos existentes y formularios y tablas para la WWW. De esta forma, el programador podrá ver rápidamente la estructura de la base de datos y acceder a ella, así como crear formularios personalizados con sólo arrastrar y soltar los campos de los mismos.

Está previsto que Visual Café para Windows salga al mercado durante este mes en versiones para Windows 95 y Windows NT y estará disponible a un precio aproximado de 50.000 pesetas..

Para más información:
http://www.symantec.com/

SQA SUITE 5
Herramienta de pruebas de aplicaciones para Windows

ADD Servicios Informáticos ha presentado en España la nueva versión de SQA Suite 5. Se trata de la primera solución de la industria “state of the art” para realizar pruebas de aplicaciones sobre Windows 3.x, Windows 95 y Windows NT. El paquete incorpora un nuevo repositorio cliente/servidor escalable en función de las necesidades del usuario.

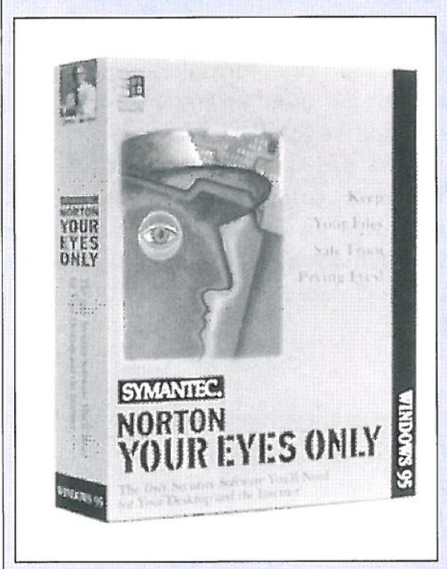
Este nuevo producto incorpora la tecnología de prueba y reconocimiento de objetos Object Testing de 16 y 32 bits para el proceso de pruebas a través de las plataformas Windows 3.x, NT y 95.

La incorporación de esta tecnología ofrece nuevas aplicaciones. Entre estas nuevas utilidades cabe destacar la posibilidad de probar aplicaciones en 16 y 32 bits utilizando un único producto y asegurar la calidad de las aplicaciones en ambientes mixtos a medida que se vayan incorporando las versiones de Windows 95 y NT dentro de la empresa.
Para más información:
ADD Servicios Informáticos.
Tel: (93) 580 25 00
Fax: (93) 580 09 95

NOVEDADES

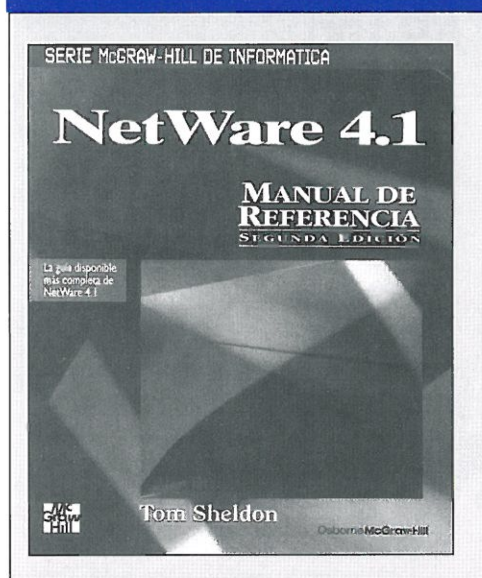
Symantec anuncia NORTON YOUR EYES ONLY para Windows 95

Symantec Corp. ha anunciado la versión 4.0 de NORTON YOUR EYES ONLY para Windows 95, un programa que proporciona protección de datos a través de características como la tecnología de encriptación instantánea (SmartLock), que decodifica los ficheros cuando los abre un usuario autorizado, y los vuelve a decodificar al cerrarlos o guardarlos. Además, con el fuerte incremento del uso de Internet para enviar información confidencial, esta característica se hace cada vez más necesaria para impedir la visualización no autorizada de la información destinada a "Norton Your Eyes Only".



Norton Your Eyes Only ofrece una solución para todo tipo de usuarios, tanto para los que utilizan portátiles como para aquellos que trabajan en red o los que están en una empresa y tienen que dejar su equipo desatendido en muchos momentos.

LIBROS



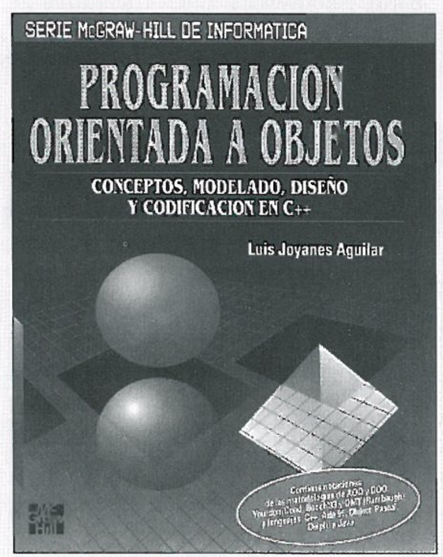
Netware 4.1 Manual de referencia. Segunda edición

Se puede calificar este libro como una completa guía de referencia para Novell Netware. El objetivo es facilitar al usuario la labor de sacar la máxima potencia a este poderoso sistema operativo de red. Esta segunda edición ha sido revisada por Tom Sheldon, ingeniero certificado por Novell, quien ha revisado y actualizado extensivamente este libro de consulta para ofrecerle la información más completa sobre Netware disponible en cualquier parte. La obra está dedicada a ususarios de Netware de cualquier nivel, que se beneficiarán sobre las nuevas características de Netware 4.1, incluyendo los Servicios de directorios de Netware, servicios de correo electrónico y sistemas de gestión de almacenamiento. Encontrará un tratamiento exhaustivo de la planificación e instalación de NetWare, interconexión de redes, gestión de los servicios de directorios y administración del sistema de archivos NetWare (NMS)

Editorial : Mc Graw-Hill
Autor : Tom Sheldon
908 páginas
Idioma : Castellano
Nivel : Todos los niveles
Precio : 6.500 (I.V.A. inc.)

Programación Orientada a objetos. Conceptos, modelado, diseño y codificación en C++

El contenido del libro se ha diseñado de modo que pueda permitir al lector ya iniciado en objetos y/o C++, y al no iniciado, adentrarse en el mundo de los objetos de un modo gradual y con la mayor eficacia posible.
La obra considera los conceptos teórico-prácticos importantes de la programación orientada a objetos, junto con los métodos correspondientes de codificación en C++.



Editorial : Mc Graw-Hill
Autor : Luis Joyanes
658 páginas
Idioma : Castellano
Nivel : Todos los niveles
Precio : 4.200 (I.V.A inc)



Como habrán observado nuestros lectores desde hace algunos números hemos insertado una página encuesta con el fin de sondear la opinión de todos los lectores sobre el contenido y el nivel de la revista.

Hemos de decir que esta experiencia ha sido muy positiva para nosotros, nos jugabamos mucho, pues nos enfrentabamos al lector cara a cara. Bueno, pues en general las calificaciones han sido muy altas, una media de 7.02 por sección nos demuestra que *Sólo Programadores* goza de buena salud. Aún así, queremos potenciar todas las secciones e incorporar alguna que vemos tienen bastante demanda por parte de los lectores. Digamos que las secciones de PC Interno, la cual deja su puesto a un nuevo curso de ensamblador, Linux, que volverá en Enero como foro abierto donde diversas plumas mostrarán sus conocimientos, y el ya finalizado Creación de un compilador, son las secciones que han tenido más éxito esta última temporada. Por detrás, pero bastante cerca de puntuación Tecnología Web, Curso de Redes, Curso de Demos y Programación básica y los lenguajes Visuales.

En cuanto a Sólo Programadores, de cara a este 1997 que tenemos a la vuelta de la esquina, comentar que, se van a ir incorporando progresivamente nuevas secciones, creemos que es más interesante para el lector que algunos cursos se conviertan en secciones, dentro de las cuales, se aborden mini cursos y también se intercalen artículos sobre algún área de interés.

A continuación la lista de los ganadores:

1. Antonio Caballero Martinez (Murcia)
2. Mercedes Tur Marti (Alicante)
3. Luis Hernan Gomez (Madrid)
4. Jose Mª Osuna (Sevilla)
5. Jose Manuel Gonzalez (Bizkaia)
6. Daniel Lázaro Cuadrado (Segovia)
7. Susana Carcamo Ruiz-Andina (Cantabria)
8. Juan Carlos Rniz García (Valencia)
9. Juan Fco.Rodriguez Martin (Madrid)
10. Felix Chamorro Atance (Madrid)
11. Felipe del Amo (Barcelona)
12. Juan Carlos Moreno Comoloa (Granada)
13. Felipe Fontoua Reguant (Barcelona)
14. Conchi Jimenez Martin (Barcelona)
15. Jose Manuel Rodriguez (Madrid)
16. Alberto Mateos Gala (Badajoz)
17. Monica Ramos García (Madrid)
18. Mª Isabel Alvarez Diaz (Madrid)
19. Ariz Muzud Tieb (Melilla)
20. Ander Lozano Perez (Bilbao)
21. Abel M.Matas García (Madrid)
22. Oscar Pascual Bakker (Tarragona)
23. Marcos Antonio (Sta. Cruz de Tenerife)
24. F. Javier Gutierrez Escacena (Sevilla)
25. Ivan Herrero Molina (Barcelona)
26. Juan Carlos García Carazo (Madrid)
27. Aurelio De la Cruz Perez (Madrid)
28. J.Carlos Baqueko Triguero (Madrid)
29. J. Ignacio Sanchon Vicente (Salamanca)
30. Fernando Perez Alonso (Madrid)
31. Jose Luis Alvarez Macias(Huelva)
32. Vicente Reiriz Godoy (Pontevedra)
33. Ignacio Hernandez Valero (Zaragoza)
34. David García Dolla (Madrid)
35. Rafael Carrion Amero (Barcelona)
36. Jesus Mª Chamizo Carmona (Madrid)
37. Juan Masana Robert (Madrid)
38. Pedro Antonio Martinez (Vizcaya)
39. Antonio Traverso Ariza (Cadiz)
40. Jose Ramon Gorchs (Malaga)
41. Alberto Vegara Cerezo (Alicante)
42. Juan Ramon Martinez Miro (Valencia)
43. Rafael Perez Sanchez (Madrid)
44. Jose Luis Triviño Rodriguez (Malaga)
45. Carlos Alvarez Diaz (Madrid)
46. Victor Manuel Juan Sanchez (Alicante)
47. Victor Espejo Gomez (Madrid)
48. Javier Orengua Miguel (Caceres)
49. Jesus Piedra Vega (Madrid)
50. Luis García Tejero (Valladolid)

CGI: METODOLOGÍA Y TRANSMISIÓN DE ESTADO

Fernando J. Echevarrieta

Como se ha presentado en anteriores artículos, la interfaz CGI es un conjunto de reglas que deben cumplir tanto los programas de aplicación como los servidores de información para establecer una pasarela entre unos y otros. Como habrán observado aquellos lectores que hayan seguido la serie, estas reglas no entrañan ninguna dificultad ni requieren de unos conocimientos especiales de programación. Sin embargo, en los primeros artículos dedicados a la interfaz, ya se anunciaba que la programación de aplicaciones CGI imponía una cierta metodología de diseño y programación, metodología que no figuraba en ninguna documentación ya que a ella, antes o después, se llegaba con la experiencia. Con el objeto de que el lector llegue antes, mejor que después, se presentan en este artículo algunas directrices a seguir a la hora de diseñar y programar aplicaciones WWW basadas en esta interfaz. No se expondrá ningún mecanismo nuevo, por lo que el artículo pretende servir también como repaso de los otros artículos dedicados a CGI, pero se plantearán nuevamente algunos mecanismos desde una nueva perspectiva, la de la transmisión de estado entre invocaciones a CGI, que es el principal problema de este tipo de programación.

La programación de aplicaciones CGI impone una metodología no descrita a la que se llega mediante la experiencia

A lo largo de los últimos meses una de las tecnologías WWW en la que más se ha profundizado es la programación CGI. Una vez expuesta la especificación, se considera ahora la metodología de programación asociada y se afronta el problema de la transmisión de estado.

PARADIGMA DE PROGRAMACIÓN CGI

Como se ha venido indicando a lo largo de la serie, el principal *handicap* de la programación CGI surge de las características del protocolo HTTP. Una vez más, recordaremos que se trata de un protocolo de transmisión sin estado. Es decir, cada conexión que se realiza desde un cliente con un servidor es siempre como la primera y no deja un "estado" que permita diferenciarla de las que se realizarán a continuación. Por otra parte, hay que recordar que a la actualización en pantalla del cliente de la información, va unida la muerte del CGI y el cierre de la conexión HTTP por lo que el estado tampoco puede ser almacenado por el script.

Así pues, a la hora de realizar una aplicación para WWW basada en la interfaz CGI, habrá que pensar en cada script CGI como un procedimiento, una función o una subrutina de la aplicación global. El problema surgirá a la hora de proporcionar una conexión entre estas subrutinas, ya que, al tratarse de programas independientes se cuenta con las siguientes limitaciones :

1. No existen variables estáticas que perduren entre una llamada y otra. En cada conexión el CGI se ejecuta

y muere quedando liberada su memoria.
2. No existen variables globales a la aplicación WWW que puedan ser compartidas por los CGIs. El usuario tiene conciencia de "aplicación" WWW, pero esto es una ilusión creada por una sucesión de CGIs, en principio, inconexos.
3. Un CGI no podrá invocar directamente a otro. La ilusión de "aplicación WWW" se produce por la sucesión de CGIs, pero estos no se pueden invocar unos a otros directamente. Lo que ocurre en realidad es que en el código HTML que generan figuran enlaces al mismo u otros CGIs.

Por tanto, cuando se desee realizar una interfaz o un programa interactivo en WWW será necesario emular estos mecanismos para poder almacenar el estado de la sesión. Así pues, a la hora de diseñar una aplicación basada en CGI, habrá que seguir los siguientes pasos :

1. Identificación de las funciones necesarias del mismo modo que se haría empleando programación estructurada.
2. Asociación de cada rutina, función o procedimiento que involucre funciones E/S (entrada o salida) a un script.
3. Diseño del método de conservación del estado y de las invocaciones a scripts.

DIVISIÓN DEL PROGRAMA EN SCRIPTS

La metodología de programación con CGIs es bastante lógica siempre que se tenga presente cómo funciona la interfaz. Así pues, una vez identificadas las funciones que debe realizar la aplicación, basta analizar una vez más este mecanismo de ejecución de CGIs para deducir cuál debe ser la partición a realizar : de cuántos scripts debe constar y qué funciones debe realizar cada uno. Recuérdese que el proceso de ejecución de CGIs es como se describe a continuación:

1. Se realiza una conexión HTTP entre cliente y servidor.

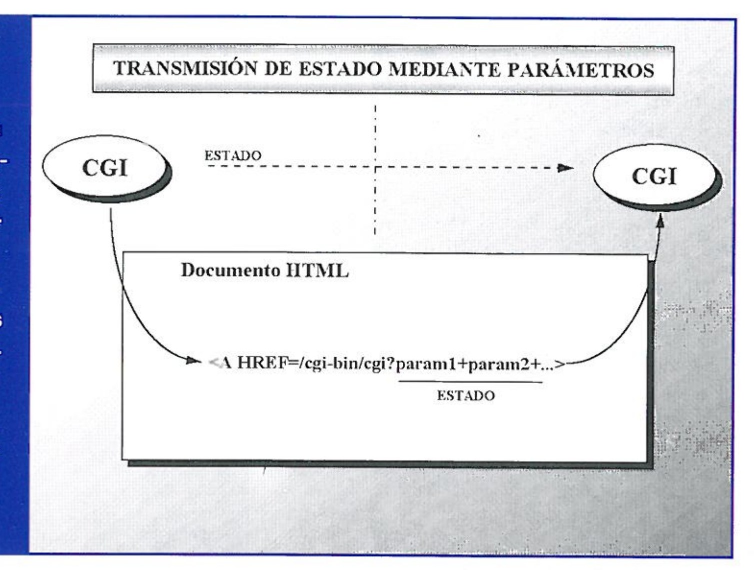


Figura 1. En la transmisión de estado mediante parámetros es posible transmitir el valor de unas pocas variables como parámetros de la siguiente invocación al CGI, incluyéndolos URL-codificados en su U

2. El servidor invoca el CGI.
3. El CGI genera datos por su salida estándar y se los envía al servidor.
4. El servidor envía la información al cliente, el CGI muere y se cierra la conexión HTTP.

Por tanto, a la hora de realizar la partición de la aplicación, las fronteras serán aquellos puntos de programa en que se realicen operaciones E/S, es decir, en los que se haga uso de la interfaz, que es precisamente el sentido de la especificación CGI. Por tanto, no es necesario realizar una ejecución de un script para cada función requerida, sino para cada intervalo entre operaciones de entrada o salida. Asimismo hay que destacar el hecho de que cada bloque entre operaciones E/S se asocia a la *ejecución* de un script, pero nada exige que deba tratarse siempre de un script diferente. Es decir, en cada uno de estos bloques se genera y muere un nuevo *proceso*, pero todos los procesos bien podrían corresponder a un mismo *programa*.

El ejemplo más sencillo de este caso, es de las famosas cámaras de vídeo conectadas a Internet. Se podría pensar en un CGI que presentara un botón que cada vez que fuera pulsado actualizara la imagen. Este CGI simplemente debería leer la nueva foto y generar una página HTML que mostrara la foto y un botón que invocara al mismo CGI. Con cada pulsación de botón se generaría un nuevo proceso que, al morir, mostraría la nueva página, pero todos los procesos corresponderían al mismo programa: el CGI que actualiza fotos.

Éste es el caso más simple, ya que, por una parte, no existe ningún estado que conservar en esta aplicación. Simplemente, se desea ver la última foto tomada por la cámara, lo que no depende de lo que haya hecho el usuario antes. Por otra parte, tampoco existe más que una función, que consiste en mostrar esta foto y proporcionar un nuevo botón para que el usuario lo pulse. Cuando el usuario lo pulsa (operación de entrada), se invoca un CGI produce una actualización (operación de salida).

Si la aplicación WWW realizara otras funciones, como inventir la matriz identidad de orden n, hallar la raíz cuadrada de 1, calcular los eclipses de luna de los próximos 45 años, consultar la bolsa de Tokio y enviarlo todo por mail a la persona que pulsó el botón, no sería nece-

La ilusión de una única aplicación WWW se consigue mediante la ejecución sucesiva de distintos scripts

sario realizar un nuevo script para cada una de estas funciones, ya que ninguna realizaría operaciones de E/S a través del WWW.

En resumen será necesaria la ejecución de un CGI cada vez que se realice una operación de E/S a través de WWW.

Es necesario tener en cuenta que cada CGI deberá siempre generar una salida, ya que, en caso contrario, la pantalla del cliente quedará en blanco y/o con un mensaje de error del tipo

Document contains no data

Por ello, en el peor de los casos, en que haya que ejecutar un CGI que no tenga que producir una salida, conviene incluir, al menos un mensaje del tipo :

Pulse aquí para continuar

Este es el típico caso de los CGIs dedicados a procesamiento de forms para envío de mensajes, donde el CGI recibe los datos tecleados por el usuario y los almacena en una base de datos o los re-envía por algún mecanismo de correo electrónico. En este caso, se suele emplear el famoso

Su mensaje ha sido enviado. Pulse aquí para continuar.

TRANSMISIÓN DE ESTADO MEDIANTE CAMPOS OCULTOS

CGI

ESTADO

CGI

Documento HTML: FORM

<INPUT TYPE=HIDDEN VALUE=param.>

ESTADO

Figura 2. En el caso de CGIs dedicados a procesamiento de forms, el estado se suele transmitir asociado a campos ocultos.

Es necesaria la ejecución de un CGI cada vez que se realice una operación de E/S a través de WWW

Esta norma, lejos de ser una restricción, permite que cada CGI genere dinámicamente la siguiente pantalla que verá el usuario, en la que puede haber invocaciones al propio u a otros CGIs y, a través del código HTML de esta pantalla se podrá realizar cierta transmisión de estado como se verá a continuación.

EL PROBLEMA DE LA TRANSMISIÓN DE ESTADO

La mayoría de las aplicaciones son interactivas y requieren consultar el estado anterior para tomar decisiones. Así, desde la aplicación más simple, por ejemplo, un programa que cuente una unidad más cada vez que se pulse un botón, hasta las más complicadas, como un juego de tablero, se necesita conocer cuales han sido las acciones seguidas hasta el momento para generar el siguiente paso. Por ejemplo, el programa que cuenta, necesita saber cuál fue el último número que contó para añadirle una unidad, no es lo mismo si fue 3, que debería devolver un 4, que si fue 7, que debería devolver 8. Un juego de tablero, va más allá, ya que necesitara conocer el estado del tablero antes de realizar o permitir un movimiento y, si es multiusuario, debera controlar también de quién es el turno.

Sin embargo, como se ha visto, en cada operación E/S un CGI nace y muere, y con él se lleva a la tumba toda su experiencia y sus conocimientos. Por ello, es necesario recurrir a mecanismos que permitan transmitir su legado, es decir que faciliten la transmisión de estado. Este es el problema fundamental de la programación de aplicaciones WWW basadas en CGI, ya que no existe ninguna indicación al respecto porque se supone que se le debe ocurrir al programador.

Realmente, es así, aunque un poco de experiencia no viene nada mal y será lo que se tratará de reflejar en el resto del artículo.

Dirigiendo la mirada a la programación tradicional, se identifican fácilmente dos formas de compartir información: mediante el paso de la misma al realizar una invocación, por ejemplo, los parámetros de una función o un programa ; o mediante el acceso a información "global" o "estática". Será necesario emular estos mecanismos a la hora de programar CGIs. Para ello, el autor ha encontrado 4 mecanismos (y agradecerá a cualquier lector que le sugiera otros nuevos):

- Uso de parámetros
- Utilización de campos *hidden*
- Acceso a memoria de masa
- Empleo de demonios guardianes de estado

USO DE PARÁMETROS

El empleo de parámetros es la forma más intuitiva de transmitir estado en una aplicación WWW. Si bien queda restringida a pequeñas cantidades de información, es la más rápida, sencilla y directa.

De la especificación de la interfaz, se recuerda que la forma de facilitar parámetros a un CGI es incluir en el URL que lo invoca el signo "cierre de interrogación" (*?*) seguido de los parámetros que se deseen separados por el signo "más" (+). Los parámetros deben encontrarse URL-codificados, es decir, que los caracteres especiales deben indicarse en hexadecimal precedidos por un signo "tanto por ciento": *%xx*. Si el CGI procesa un form, deberá URL-descodificar la información que se le envía [Echeva-1] pero si no, la información le llegará ya URL-descodificada en forma de parámetros.

Nada mejor como ejemplo que ella citado del programa que cuenta. En el listado 1 se puede ver un CGI en sh que realiza esta función y en el listado 2, el mismo programa escrito en C. Como se ve, son extremadamente simples, simplemente, toman un parámetro que incrementan. Lo muestran y mueren. Pero junto a él generan un botón para su próxima invocación y, aunque el usuario no percibe la diferencia, en el URL de la invocación dejan el estado de la cuenta antes de morir, por lo que se ha conseguido el objetivo buscado.

Un ejemplo igual de simple pero algo más útil es la realización de un CGI que recorra un árbol de directorios. En el listado 3 se puede observar el código en sh del mismo. La idea es la misma, simplemente debe indicar a qué directorio ha cambiado el usuario. Así, el script muestra el contenido del directorio "actual", generando un enlace a sí mismo por cada uno de los subdirectorios. Este enlace contiene un parámetro distinto para cada subdirectorio que indicará a la nueva instancia del CGI qué directorio debe mostrar. De esta forma, aunque el usuario cree utilizar una única conexión con el servidor y ejecutar un único proceso que recorre directorios, cada vez que hace click con su ratón se genera y muere una nueva conexión HTTP y se invoca y muere un nuevo proceso CGI.

LISTADO 1 : CGICUENTA.SH

```
#!/bin/sh
COUNTER=$1
if [ "$COUNTER" = "" ]; then
  COUNTER=0
fi
COUNTER=`expr $COUNTER + 1`
echo Content-type: text/html
echo ""
echo "CUENTO:$COUNTER<p>"
echo "<A HREF=/cgi-bin/CGIcuenta.sh?$COUNTER>"
echo "Pulsa aqui para seguir contando"
echo "</A>"
```

LISTADO 2 : CGICUENTA.C

```
void main(int argc, char *argv[])
{
        int counter;

        if (argc == 2)
      counter = atoi(argv[1]);
    else
      counter = 0;

        printf("Content-type: text/html\n\n");
        printf("CUENTO:%d<p>",counter);
        printf("<A HREF=/cgi-bin/CGIcuenta?%d>",++counter);
    printf("Pulsa aqui para seguir contando");
        printf("</A>");
}
```

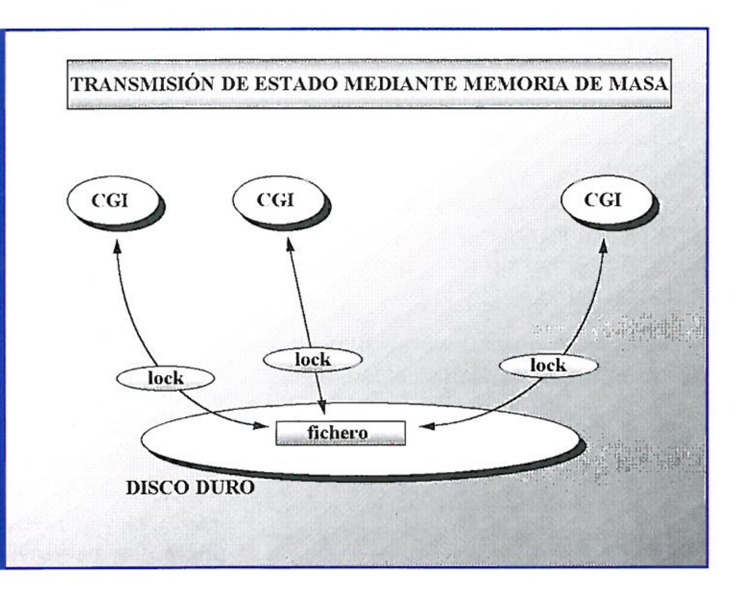


Figura 3. Cuando la cantidad de información a transmitir alcanza cierto volumen, o debe almacenarse de forma estable se debe recurrir al disco duro. En este caso conviene gestionar el acceso concurrente, por ejemplo, mediante el uso de locks.

LISTADO 3 : CGIDIRECTORY

```
#!/bin/sh

echo Content-type: text/html
echo

cd $1
for i in *
do
  if [ -d $i ]; then
echo  "<A  HREF=/cgi-bin/CGIdirectory?`pwd`/$i>$i</A><P>"
  else echo "$i<P>"
  fi
done
```

UTILIZACIÓN DE CAMPOS HIDDEN

Con el procesamiento de forms, con el que no se deben enviar parámetros al CGI encargado de procesar el form (estudio que se realizó en [Echeva-2]), el mecanismo anterior pierde su validez. La idea es que cuando un CGI genera una serie de formularios de forma consecutiva, se pueda transmitir información de estado a la siguiente ejecución del CGI y, ya que no se puede hacer a traves de parámetros, se realice a través de un campo de entrada de datos <INPUT> con valor predefinido. El problema surge de que el usuario lo vería (no como en el caso de los parámetros, que figuran en el URL del enlace) y, lo que es peor, podría modificarlo. Es por ello, por lo que se introdujo en HTML 2.0 el tipo HIDDEN para el atributo TYPE de la etiqueta <INPUT>. De esta forma, se hace posible incluir campos con valor preestablecido por el CGI que genera la página y que el usuario no vera en pantalla ni podrá modificar. Cuando se produzca el envío, el nuevo CGI recibirá esta información como si de un campo más se tratara, con lo que se habrá conseguido nuevamente la transmisión de estado. Este proceso fue ya descrito en [Echeva-1], aunque se recuerda aquí por completitud y con el fin de permitir una mejor asimilación estudiándolo en este entorno.

Como ejemplo, imagine el lector una encuesta en que se solicita el nombre del encuestado a través de diferentes forms en el primero de los cuales se le solicita el nombre. El código HTML correspondiente a este form podría ser :

<FORM METHOD=POST ACTION=/cgi-bin/form1>
Escribe aquí tu nombre:
<INPUT TYPE=TEXT NAME=name>
</FORM>

Si después se desea que este nombre encabece cada uno de los forms, será necesario transmitirlo al CGI encargado de generar el siguente form. Así, el script *form1* invocado en el ejemplo, deberá generar una nueva página HTML formulario con un campo HIDDEN que contenga el nombre tecleado (por ejemplo, echeva):
<FORM METHOD=POST ACTION=/cgi-bin/form2-cgi>
<INPUT TYPE=HIDDEN NAME=name VALUE=echeva>

(... resto del form ...)
</FORM>

Cuando se ejecute el CGI *form2* recibirá, como un campo más, el nombre *echeva* tecleado en una fase anterior, con lo que nuevamente se habrá conseguido el objetivo de realizar una transmisión de estado.

LISTADO 4

```
void main(int argc, char *argv[])
{
        switch (atoi(argv[1]) {
                case 1 : ajedrez1() ; break ;
                case 2 : ajedrez2() ; break ;
                case 3 : ajedrez3() ; break ;
                (...)
                case   n   :   ajedrezn()   ;
break ;}
```

UNA NOTA SOBRE LA DIVISIÓN EN PROGRAMAS

Como se puede intuir, si una aplicación WWW realiza numerosas operaciones de E/S atendiendo a diferentes funciones, el número de scripts CGI necesarios para una única aplicación puede crecer considerablemente. Algunos programadores realizan únicamente un CGI por cada aplicación WWW empleando un parámetro para identificar la función que este debe realizar. Así, sería equivalente disponer de un conjunto de aplicaciones *ajedrez1 ajedrez2 ajedrez3 ... ajedrezn* para implementar un juego de ajedrez, o emplear una única *ajedrez* que aceptara un parámetro que pudiera tomar los valores *1, 2, 3, ..., n* y, de esta forma, determinara su comportamiento. Cada opción tiene sus ventajas y sus inconvenientes. Así, en el primer caso, el directorio de CGIs pronto se vera repleto. En el segundo, habrá que concatenar todos los scripts en un único fichero fuente más largo, lo que lo hará más dificilmente legible. Recuérdese que cada una de las funciones de este script será independiente de las demás. El mecanismo de programación es el mismo, sólo se trata de almacenar el código en uno sólo o varios ficheros. Así, si el programa de ajedrez, constara de una serie de scripts en C, se podrían unificar en un único fichero cuya función main sería algo como el listado 4:

También hay que señalar que, en algunos casos, el número de parámetros crece considerablemente y al encontrarse URL-codificados, es difícil distinguir los parámetros a simple vista observando el URL que invoca al script. Y recuerde el lector, que la primera invocación a un CGI deberá escibirse manualmente en una página HTML.

ACCESO A MEMORIA DE MASA

Como *memoria de masa* se entienden aquellos sistemas de almacenamiento estable de información, lo que en la práctica equivale a hablar del disco duro. Este mecanismo equivaldría al del empleo de las variables globales o estáticas en la "programación habitual". Así, en algunas ocasiones, el empleo de parámetros o campos HIDDEN se hace del todo insuficiente. Por ejemplo, pensemos en el caso de un contador de accesos. Este tipo de contador es muy diferente del empleado como ejemplo en un apartado anterior, ya que la información de los accesos debe permanecer estable aún cuando el usuario haya abandonado la página que invoca el CGI. Todos los usuarios deberán incrementar y el mismo valor y no llevar cada uno una cuenta particular que comience siempre desde el mismo valor. Así pues, será necesario almacenar en disco el valor a incrementar con cada acceso.

El listado 5 muestra un ejemplo del empleo de esta técnica para la actualización de un contador.

PELIGRO: PROGRAMACIÓN CONCURRENTE

El ejemplo del contador ya se empleó en [Echeva-3] y por ello, puede que algun lector se haya echado las manos a la cabeza al ver el listado 5. Por su importancia y para aquellos lectores que no dispongan de aquel artículo , se revisitará el problema.

LISTADO 5

```
#!/bin/sh
COUNTER=`cat $FILE`
if [ "$COUNTER" = "" ]; then
   COUNTER=0
fi
COUNTER=`expr $COUNTER + 1`
echo $COUNTER > $FILE
echo Content-type: text/html
echo ""
echo $COUNTER
```

Ciertamente, el listado 5 es un programa incorrecto. Al programar CGI hay que tener siempre en cuenta al emplear un medio externo para almacenar información, que puede haber otro proceso que acceda a la vez a esa misma información almacenada. Es decir se trabaja en un entorno de programación concurrente. El estudio de la programación concurrente es toda una disciplina que, por supuesto, no se puede tratar en este artículo. Sin embargo, se señalaran dos problemas que debe tener encuenta todo programador de CGIs que emplee medios externos para transmitir estado:

1. El (posible) acceso simultáneo de dos o más procesos a la misma información.
2. La (posible) utilización simultánea de un mismo recurso por parte de dos o más procesos para almacenar información de estado.

En general, el autor aconseja al programador de CGIs que siempre que

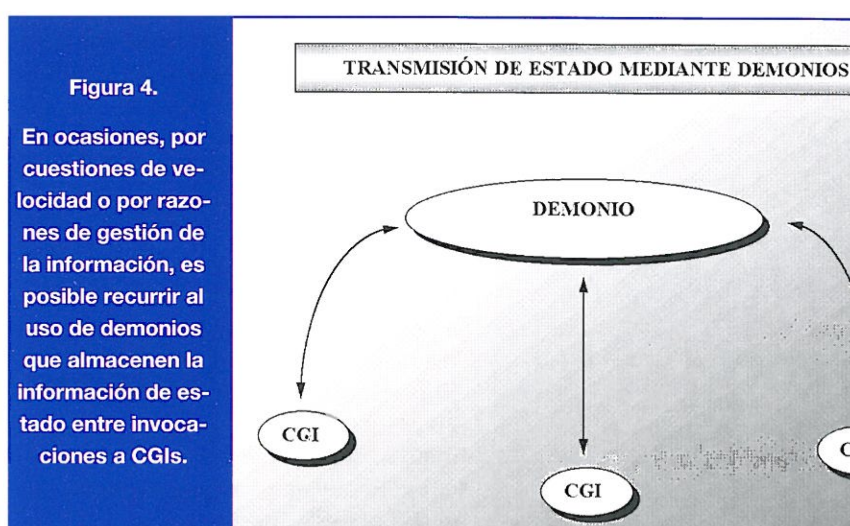


Figura 4. En ocasiones, por cuestiones de velocidad o por razones de gestión de la información, es posible recurrir al uso de demonios que almacenen la información de estado entre invocaciones a CGIs.

realice un acceso sobre un medio externo, como el disco, se plantee la pregunta : ¿Qué pasaría si otro usuario arranca el mismo CGI e intenta realizar una operación sobre este elemento externo a la vez que lo hace este proceso en el que me estoy fijando?

Por ejemplo, si el script lee el valor del fichero (por ejemplo contador=5), le suma uno en su memoria (contador=6) y antes de que le dé tiempo a actualizar el fichero otro script lee también el valor del fichero (contador=5) y le suma 1 en su memoria (contador=6), ya, independientemente del orden en que escriban el valor del contador en el fichero, éste contendrá un valor de 6, cuando, en realidad, debería contener un 7 porque ha habido 2 nuevos accesos. En CGI, este problema se podrá resolver siempre mediante el empleo de locks, o cerrojos que se activen cuando un proceso accede al elemento compartido. Estos locks pueden ser simplemente, la existencia o no de un fichero vacío. El listado 6 muestra el problema corregido mediante el empleo de locks.

LISTADO 6

```
#!/bin/sh

LOCK=/tmp/.counter_lock
FILE=/tmp/.counter_file

while [ -f "$LOCK" ]
do
  sleep 1
done

# Genera cerrojo
touch $LOCK

# Incrementa contador
COUNTER=`cat $FILE`
if [ "$COUNTER" = "" ]; then
  COUNTER=0
fi
COUNTER=`expr $COUNTER + 1`
echo $COUNTER > $FILE

# Libera cerrojo
rm $LOCK

# Salida del CGI
echo Content-type: text/html
echo ""
echo $COUNTER
```

EMPLEO DE DEMONIOS GUARDIANES DE ESTADO

Aún así, en ocasiones, el empleo del disco se vuelve más complejo. Es el caso 2 que se mencionaba en los párrafos anteriores. Imagine el lector, por ejemplo, la aplicación de ajedrez de la que se hablaba anteriormente. Esta claro que la situación del tablero es el estado que se debe conservar. Pero es demasiado grande para transmitirla como parámetros. Así pues, la primera idea es almacenarla en disco. Cada CGI que se generara en cada movimiento, debería leer todo el tablero del disco, realizar las modificaciones oportunas y volver a almacenar el tablero completo nuevamente en disco. Los puristas podrían protestar. ¡El acceso a disco es dos órdenes de magnitud más lento! Realmente, esto no importa dado el tamaño de un tablero, lo que haría que prácticamente siempre se encontrara en la caché de disco del S.O. Algún lector tras leer los párrafos anteriores pensará : "¿qué pasaría si tuviera 23 personas jugando al ajedrez a la vez en mi servidor?". A lo mejor ya no se encontrarían los datos en caché. Tampoco ese es problema, dadas las labores que debe realizar el servidor, el tiempo de transmisión y la latencia de la red. Pero la pregunta de ese lector daría en el clavo : al utilizar el mismo recurso compartido el tablero del jugador j se grabaría encima del jugador i. Ahora la utilización de locks no es buena solución a no ser que cree un lock al iniciar la partida y se cierre al terminarla, pero, al tratarse de ajedrez, es posible que el segundo jugador se inquietara "algo" al esperar. Una posible solución podría ser emplear un fichero diferente para cada tablero, pero seria necesario gestionar los nombres de los ficheros de alguna manera y habría que acceder constantemente al sistema de ficheros. Quizá sería más elegante poder realizar todo esto empleando memoria dinámica. Sin embargo, la memoria de cada CGI se liberará cuando el CGI muera.

Lo ideal sería poder disponer de un mecanismo externo, flotando en la memoria del ordenador, que almacenara toda la información de estado que se deseara y no muriera con el CGI. Lo que se está definiendo es un demonio. Este demonio actuaría como aplicación servidora y cada CGI que accediera al estado que salvaguardara el demonio sería cliente del mismo. Aquellos lectores que no conozcan o deseen profundizar en los mecanismos de programación de aplicaciones cliente-servidor, podrán encontrar toda la información necesaria en [Echeva-4] y [Echeva-5].

CONCLUSIONES

Por supuesto, la programación de demonios no es objeto de esta serie, del mismo modo que no lo es el lenguaje C, ni la sh, ni se han tratado los mecanismos de acceso al sistema de ficheros. En cualquier caso, para aquel programador que conozca todas estas técnicas, es necesario mencionar cuando es adecuado utilizarlas.

Lo importante es conocer a priori cuales son los problemas que van a surgir a la hora de realizar una plicación WWW basada en CGI, realizar una partición eficiente y elegir la mejor forma que los conocimientos del programador, o el propio sistema operativo, permitan. Bien es cierto, que esta metodología se adquiere con la experiencia, aunque el autor espera que este artículo sirva, al menos, como guía. Ahora es labor del lector imaginar una aplicación y hacerla accesible a través de WWW. Piense por ejemplo en un servicio de consulta a una base de datos, o un juego de tablero, o...

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica en relación con el artículo, se anima al lector a que se ponga en contacto con el autor a través de :

E-mail Internet: echeva@dit.upm.es
E-mail CompuServe: 100646,2456
WWW : http ://highland.dit.upm.es:8000

REFERENCIAS

Todas las referencias corresponden a artículos del autor publicados en "Sólo Programadores" según :

[Echeva-1], "CGI: Envío de información a un servidor", núm. 21
[Echeva-2], "La interfaz CGI : descripción avanzada", núm. 25
[Echeva-3], "La interfaz CGI : primer contacto", núm. 20
[Echeva-4], "La interfaz socket (I)", núm. 17
[Echeva-5], "La interfaz socket (II)", núm. 18

ARCHIVOS INDEXADOS (I)

José C. Remiro

Una gran parte de los programas que se desarrollan están dedicados al tratamiento de grandes volúmenes de información almacenada de forma permanente en los dispositivos de almacenamiento secundario (discos, streamers, disquetes, etc.). Este artículo mostrará una panorámica de cómo se almacena dicha información y los métodos disponibles para el tratamiento eficaz de ésta.

Todos los lenguajes de programación de alto nivel permiten tratar la información almacenada en los dispositivos de almacenamiento secundario utilizando el concepto de archivo: un conjunto de datos que posee la misma estructura y están relacionados. Este concepto es un modelo respecto de la forma en que se almacena la información en los dispositivos físicos. Así, los métodos para almacenar la información en un disco y en una cinta magnética son totalmente diferentes, pero a través de los sistemas operativos que definen la forma en que los dispositivos gestionan los archivos, los métodos de acceso y asignación, así como la estructura de directorio, es posible la utilización de archivos de una forma independiente del medio en que éstos se almacenan. Además, esta abstracción permite que los lenguajes y metodologías de programación puedan trabajar sobre diferentes sistemas operativos sin apenas modificaciones. De esta forma el código de un programa para un determinado sistema, al menos en teoría, serviría para otro sistema totalmente diferente.

La forma en que está organizada la información en un archivo no es suficiente, normalmente, para acceder a ella de forma eficaz. En esta entrega se describirá cómo se puede resolver este problema, en resumidas cuentas estudiaremos el concepto de archivo.

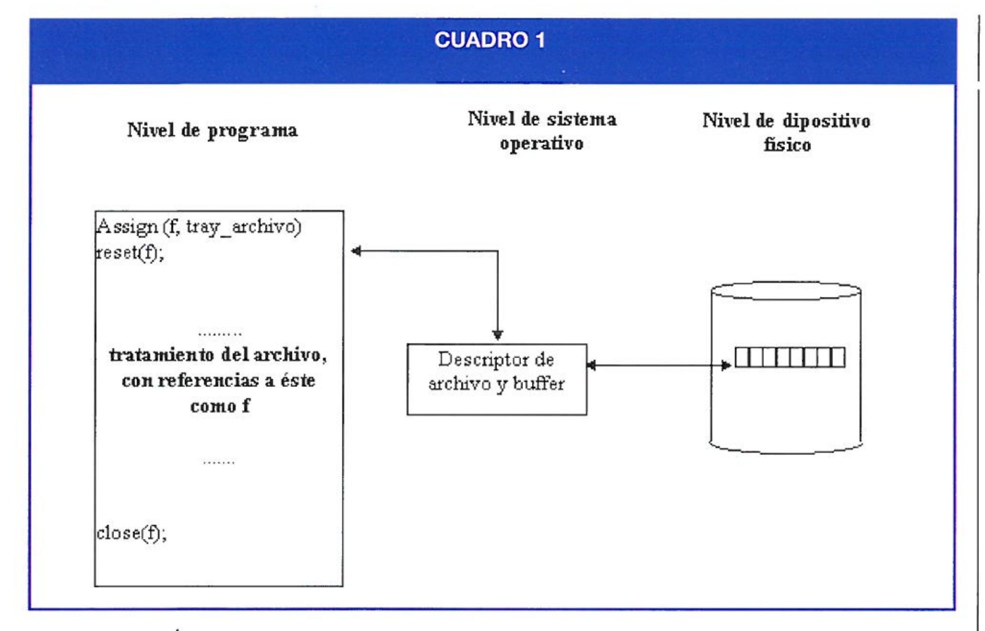
ARCHIVOS FÍSICOS Y LÓGICOS

Todos los lenguajes de programación tratan los archivos físicos, colección de datos relacionados almacenados en un dispositivo de almacenamiento secundario, como si se tratasen de archivos lógicos. A nivel de programa no interesa conocer la forma física en que se almacena la información, si ésta existe, cómo se denomina y las propiedades que posee. Todo ello se deja al sistema operativo (cuadro 1). Al programador sólo le interesa la estructura de la información que puede albergar el archivo, organizada en estructuras de alto nivel, normalmente registros. Así, en cualquier lenguaje se puede declarar un archivo como si se tratara de una variable más, dotada de su correspondiente estructura de datos y a la que se puede aplicar una serie de procedimientos y funciones destinadas a manipularlos. Por ejemplo, en Turbo Pascal, para declarar de forma lógica un archivo se utiliza la sección dedicada a la declaración de variables, siendo su tipo File y a continuación la estructura asociada al archivo.

Ahora bien, los lenguajes de programación también deben procurar alguna forma para indicar que el archivo declarado en un programa se encuentra físicamente en algún lugar. En Turbo Pascal existe el procedimiento predefinido *Assign(tipo_fichero,nombre_fichero)*. Este procedimiento tiene como primer parámetro el nombre de archivo lógico, es decir, aquel que va a referenciar al archivo físico dentro del programa. Junto con este parámetro se encuentra una cadena de caracteres que indica un nombre de archivo a nivel del sistema operativo. Por ejemplo, la siguiente llamada:

Assign(arch_clientes,'c:\maestro\client.dat')

indicaría que cada vez que el programa hiciese referencia a arch_clientes se estaría realizando una operación



sobre el archivo físico cuya trayectoria es c:\maestro\client.dat. El anterior procedimiento no hace más que reservar un bloque de información de archivo (FIB, File information block) en la memoria central que todo archivo debe poseer para intercambiar información con el programa. Además, se reserva un buffer donde realmente se leen y escriben de forma intermedia los datos hacia/desde el archivo, pues debido a la lentitud de la transferencia de información de los dispositivos de almacenamiento (respecto de la velocidad de acceso de la memoria central) se espera que este buffer esté lleno para realizar las operaciones de transferencia para así poder utilizar el canal de transferencia en su totalidad.

Esta forma de trabajo permite poder utilizar el programa solamente modificando la trayectoria del archivo físico dentro del programa si ésta cambia, o bien, utilizar el mismo programa sobre otro archivo distinto que disponga de la misma estructura que el anterior.

La tendencia de los lenguajes de programación más modernos es intentar que el código del programa sea independiente de la estructura de los archivos, debido a que cualquier cambio sobre la estructura repercutiría de forma inmediata en el mantenimiento del programa. Aunque esto no es del todo posible, sí que se han construido lenguajes de manipulación y consulta de bases de datos, como SQL, que permiten realizar programas de una forma más rápida y casi independientemente de la estructura de la información, además de aislar completamente el programa del entorno físico donde éste se ejecuta. Todo lo que sigue a continuación se referirá a lenguajes de programación de propósito general.

Un archivo es un conjunto de datos que posee la misma estructura y están relacionados

MODOS DE APERTURA Y MÉTODOS DE ACCESO

Para conseguir que un archivo quede preparado para realizar operaciones de lectura y/o escritura debe ser abierto. Un archivo abierto tiene en su bloque de información de archivo un indicador que apunta al registro del archivo que va a ser tratado a continuación. Se tomarán como modelo los modos de apertura de Turbo Pascal, *reset(nom_arch) y rewrite(nom_arch)*.

El primer modo de apertura sitúa el indicador en el primer registro del archivo (numerado por 0), quedando el contenido del archivo intacto, para a partir de él comenzar el tratamiento a realizar. Si no existiese el archivo, este modo de apertura provocaría un error. En el segundo modo de apertura, si el archivo no existe físicamente, se crea y si existe, se elimina su contenido quedando el indicador de registro en la primera posición (numerada por 0).

Una vez que se ha realizado el procesamiento de los archivos hay que proceder a cerrarlos. Esto obliga a que los posibles registros que se encuentran almacenados en el buffer de intercambio y que aún no han sido trasladados al dispositivo de almacenamiento secundario sean grabados. Aunque en algunos textos se indique que en ocasiones no es necesario cerrar un archivo (por ejemplo, cuando éste no ha sido modificado) esta decisión puede provocar que el archivo se corrompa. Además, el hecho de no cerrar un archivo indica que éste tendrá asignado un descriptor de archivo que puede ser necesario para otro. El procedimiento correspondiente en TurboPascal es *close(nom_arch)*.

Las operaciones básicas asociadas a un archivo son la lectura y la escritura, unidas a la detección del final del archivo. La primera provoca que la información almacenada, señalada por la posición del indicador de registro, pase a ser el contenido de la variable indicada en el procedimiento de lectura (evidentemente, ésta debe ser del mismo tipo que el registro del archivo). Además, el indicador de registro pasará a señalar al siguiente registro. La operación de escritura provoca la actualización o la creación de un nuevo registro (solamente en la posición final) en el archivo, justo en la posición señalada por el indicador de registro. Así, esta operación sirve bien para sobreescribir la información de un registro que ya existía o bien para insertar nuevos registros al final del archivo. Además, el indicador de registro del archivo pasa a señalar al siguiente registro. Por último, se considerará una función booleana que aplicada al archivo devolverá el valor verdadero si el indicador de registro se encuentra situado al final de éste. La función correspondiente en Turbo Pascal es *eof(nom_arch)*, que devuelve verdadero si se ha alcanzado el final del archivo.

Por lo descrito hasta ahora, si sólo se dispone de estas operaciones y de los dos modos de apertura, únicamente se puede realizar un tratamiento secuencial de los archivos, pues si se abre un archivo con el procedimiento reset sólo se podrá pasar de un registro a otro reescribiendo los registros que ya existían o leyéndolos, pudiendo incluir nuevos registros al detectar el final del archivo. Si el modo de apertura seleccionado es el correspondiente al procedimiento *rewrite,* entonces la única operación admisible es la creación de nuevos registros en el archivo mediante la operación de escritura y en el orden en que son escritos.

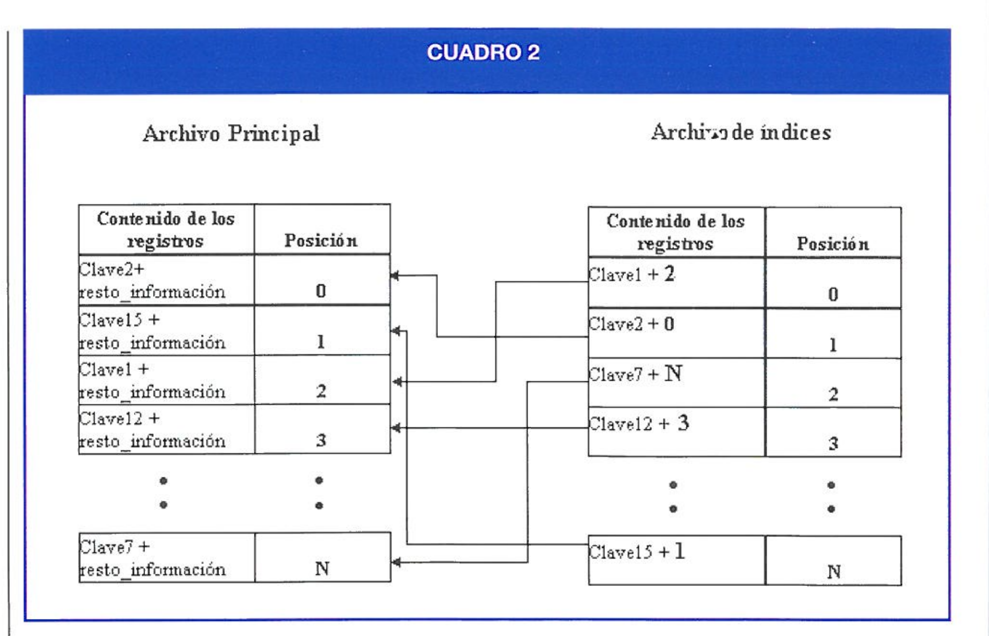
Cuando un archivo se utiliza de la forma anterior se dice que se está accediendo a él secuencialmente, pues para recuperar la información almacenada en un registro es necesario haber accedido a los registros que le preceden. Este método de acceso es deseable cuando se dispone de un dispositivo de almacenamiento secundario de acceso secuencial, como puede ser una cinta magnética, pero si se desea trabajar con dispositivos de acceso aleatorio el tratamiento secuencial de un archivo es inadmisible, estando reservado para casos muy especiales como puede ser la realización de una copia de seguridad o de un proceso batch en el que deban ser tratados la totalidad de los registros y el orden en que fueron grabados sea el adecuado.

Todos los lenguajes de programación tratan los archivos físicos como si se tratasen de archivos lógicos

En efecto, existe una instrucción, a la que se denominará posicionar, que permite modificar el indicador de registro. De esta forma, para poder recuperar o reescribir un registro basta con aplicar dicha instrucción junto con la posición que ocupa el registro dentro del archivo y ejecutar la instrucción de lectura o escritura correspondiente, teniendo cuidado que la posición indicada se encuentre entre 0 y el número de registros del archivo. Con esta instrucción es posible el tratamiento de un archivo con acceso a sus registros de forma directa. De esta forma, si se desea modificar un registro basta con posicionar el indicador de registro sobre él y realizar una operación de escritura.

En Turbo Pascal, los procedimientos de lectura y escritura tienen la forma *read(nom_arch,registro)* y *write(nom_arch,registro)* respectivamente, siendo registro una variable del mismo tipo que el registro asociado al archivo y donde se realizará el intercambio de información. El procedimiento de posicionamiento se corresponde con *seek(nom_arch,posicion),* donde posición es una variable de tipo entero largo. Además, existen dos funciones que pueden ayudar a acceder a un archivo de forma directa: *filePos(nom_arch),* que devuelve la posición actual a la que apunta el indicador de registro y *filesize(nom_arch),* que devuelve el número total de registros de que dispone el archivo. El parámetro numérico del procedimiento seek siempre debe encontrarse entre 0 y el valor devuelto por filesize.

CUADRO 2



ARCHIVOS DE ÍNDICES

La gestión de la información de un archivo se ve mejorada con el acceso directo, pero aún existen problemas si ésta se desea utilizar sin más. En principio, no hay ninguna relación entre la información almacenada en el registro y la posición que ocupa en el archivo. Por tanto, si se desea localizar un registro que incluya una determinada información, de nuevo habrá que buscar a lo largo y ancho del archivo. Piénsese en la siguiente situación: si se dispone de un archivo para almacenar los datos acerca de clientes y se les asigna como clave para identificarlos la posición del registro donde se almacenan sus datos, sería muy fácil recuperar los datos conocida su clave, pero la búsqueda debería ser secuencial si se desea realizar una consulta conocido el nombre del cliente. Esta situación se complica aún más cuando se desea eliminar el registro de un cliente, pues para mantener el resto de las claves intactas habrá que dejar un registro inutilizado en el archivo, y si esta operación se realiza frecuentemente podría darse el caso de que el número de registros utilizados fuese inferior al de los no utilizados.

Una de las posibles soluciones para resolver el problema anterior consiste en realizar un archivo índice. Tómese como referencia el índice de un libro, donde se pueden encontrar los términos más usuales por los que el usuario puede buscar y junto a ellos el número

de la página donde se hace referencia a ellos. La utilidad de este índice reside en que la búsqueda es muy rápida, pues los términos están ordenados alfabéticamente y la mecánica para acceder a la totalidad de la información es muy sencilla: primero buscar el término en el índice y posteriormente obtener la información dirigiéndose a la página asociada al término.

Un archivo índice no es más que un archivo formado por pares (clave,dirección). La clave es un campo o una combinación de ellos por la que se identifica uno o varios registros y la dirección es la posición que ocupa el registro que guarda la totalidad de la información, situado en otro archivo denominado principal (cuadro 2). De esta forma, y siguiendo con la analogía del libro, las páginas de éste serían los registros del archivo principal, mientras que cada uno de los pares del archivo índice serían los términos junto con la dirección de la página.

Existen varias restricciones a la hora de realizar un archivo índice, puesto que para localizar un registro del archivo principal habrá que realizar dos accesos: el primero para determinar en el archivo índice la posición que ocupa el registro en el archivo principal y posteriormente el acceso a este último. Esto implica que la búsqueda en el archivo índice debe ser lo más rápida posible. Como consecuencia de esto el archivo debe estar ordenado y el algoritmo de búsqueda debe ser eficiente.

También existen problemas con la consistencia de los datos, pues cualquier eliminación o inserción dentro del archivo principal debe ser reflejada en el archivo índice, este problema aumenta si el archivo principal tiene varios archivos índices, pues la eliminación de un registro provoca inmediatamente la necesidad de eliminar cada uno de los registros de los archivos índice que hacían referencia al registro eliminado. Para resolver este problema, hay que realizar una gestión añadida recurriendo a lo que se denomina borrado lógico. Esta técnica consiste en añadir un campo booleano extra a los registros, no visible para el usuario de la información, de tal forma que cuando éste desee eliminar un registro se asigne un valor al campo convenido por el programador, indicando de esta forma que dicho registro ha sido eliminado. Además, para evitar la acumulación de los registros marcados para su eliminación, hay que realizar un proceso periódico para borrarlos definitivamente teniendo que ser reconstruidos tanto el archivo principal como sus archivos índices.

También a la hora de planificar un programa que utilice archivos índice habrá que tener en cuenta los diferentes tipos de claves que puede haber. Así, se hablará de clave primaria si conocida ésta identifica de manera unívoca a un registro. Por tanto, si se desea construir un archivo índice de claves primarias habrá que asegurar, para salvaguardar la consistencia, que todos los registros de éste índice son distintos.

Las operaciones básicas asociadas a un archivo son la lectura, la escritura y la detección del final de éste

EJEMPLOS

En el CD-ROM que acompaña a la revista se pueden encontrar dos units y un programa que hace uso de ellas para realizar las altas en un archivo que dispone de un índice para una clave primaria.

Se supone que se desea construir un archivo que contenga una palabra técnica, una descripción de ésta y el concepto al que está asociada. Esta información estaría almacenada en el archivo principal. Supóngase que los usuarios están interesados en acceder a la anterior información indicando la palabra. Es evidente que ésta constituye una clave primaria del archivo principal, por lo que se ha construido el correspondiente archivo índice, constituido por todas las palabras que contiene el archivo principal y la posición que ocupan en aquel.

Cuando un usuario desea incluir una nueva palabra se comprueba que ésta no se encuentra ya en el archivo principal inspeccionando el archivo de índices. Este proceso se realiza mediante una búsqueda por bipartición (descrito en anteriores artículos). Ya que el índice se encuentra ordenado ascendentemente según el código ASCII y es posible conocer el número de registros de que dispone el archivo (utilizando la función *filesize*), se puede aplicar el anterior algoritmo de búsqueda. Pero este proceso tiene un pequeño inconveniente: los registros que se han introducido con anterioridad no están representados aún en el archivo de índices, han sido insertados en el archivo principal y se ha insertado en una lista auxiliar su clave y la posición en la que han sido insertados (el último registro del archivo principal en el momento de ser insertados). Esto se ha hecho así debido a que el archivo de índices debe estar ordenado y, por tanto, la inserción en dicho archivo hubiera provocado su modificación total, consumiendo este proceso gran cantidad de tiempo y causando una espera innecesaria al usuario.

Por tanto, la comprobación de si una palabra se encuentra o no en el archivo se realiza sobre el archivo de índices (la palabra fue insertada en una sesión anterior) o bien en una lista que se mantiene ordenada y que va creciendo cada vez que el usuario introduce un nuevo registro.

Puede ocurrir que una palabra introducida con anterioridad fuese borrada, es evidente que no se puede eliminar directamente del archivo principal a menos que sean tratados todos sus registros, copiando aquellos que deban permanecer en un archivo auxiliar e ignorando el registro que se desea borrar y posteriormente renombrando el archivo auxiliar con el nombre del principal, proceso a todas luces ineficiente. Así, al registro del archivo de índices y a los nodos de la lista que representa a los registros actualmente insertados, se les ha dotado de un campo adicional (con nombre borrado y tipo booleano) para indicar que ha sido marcado y así

CUADRO 3

```
procedure fusiona_arch_lista (var f: arch_indice; var l: lista);
 var
  f_aux: arch_indice;
  l_aux: lista;
  reg_aux: reg_indice;
 begin
  assign(f_aux, 'temporal.dat');
  rewrite(f_aux);
  l_aux:= l;
  if filesize(f) = 0 {el archivo está vacio}
   then
    while (l_aux <> nil)
       do
        begin
         write(f_aux, l_aux^.contenido);
         l_aux:= l_aux^.sig
        end
   else
    begin
     seek(f,0);
     read(f, reg_aux);
     while (l_aux <> nil) and (not eof(f)) {mezcla de ambos}
        do
         if l_aux^.contenido.clv < reg_aux.clv
          then
           begin
            write(f_aux, l_aux^.contenido);
            l_aux:= l_aux^.sig
           end
          else
           begin
            write(f_aux, reg_aux);
            read(f, reg_aux)
           end;
     if (not eof(f)) {no hay mas nodos en la lista}
       then
        begin
         while (not eof(f))
            do
             begin
              write(f_aux, reg_aux);
              read(f, reg_aux)
             end;
         write(f_aux, reg_aux)
        end
       else {hay nodos en la lista y un registro pendiente}
        begin
         while (l_aux <> nil) and (l_aux^.contenido.clv < reg_aux.clv)
           do
            begin
             write(f_aux, l_aux^.contenido);
             l_aux:= l_aux^.sig
            end;
         write(f_aux, reg_aux);
         while (l_aux <> nil)
            do
             begin
              write(f_aux, l_aux^.contenido);
              l_aux:= l_aux^.sig
             end;
        end;
    end;
  close(f_aux);
  close(f);
  erase(f);
  rename(f_aux,tray_arch_ind);
 end;
```

poder eliminarlo posteriormente. Esta decisión en el diseño de los archivos índice provoca la siguiente situación: supóngase que existe un registro marcado como borrado, si se desea introducir un nuevo registro con la misma clave se debería permitir realizar la inserción, pero como los procesos de búsqueda devuelven el primer registro encontrado dada su clave, no sería conveniente introducir un nuevo registro, sino introducir la nueva información en el archivo principal sobre la posición indicada por el archivo índice (o la lista auxiliar) estableciendo el valor del campo borrado como true, pues el registro que anteriormente estaba marcado como borrado ahora contiene la información asociada a la misma clave.

Como tarea final a realizar se encuentra la actualización del índice para incluir los elementos que están almacenados en la lista, correspondientes a los registros que se han introducido durante la ejecución del programa altas. Puesto que el archivo de índices está ordenado y la lista también, se ha realizado un proceso de fusión entre el archivo y la lista sobre un archivo auxiliar donde, alternativamente y dependiendo del valor de la clave, se van escribiendo. Posteriormente, basta con cerrar ambos archivos y renombrar el archivo resultante de la fusión, siendo este proceso un ejemplo de acceso a un archivo de forma secuencial.

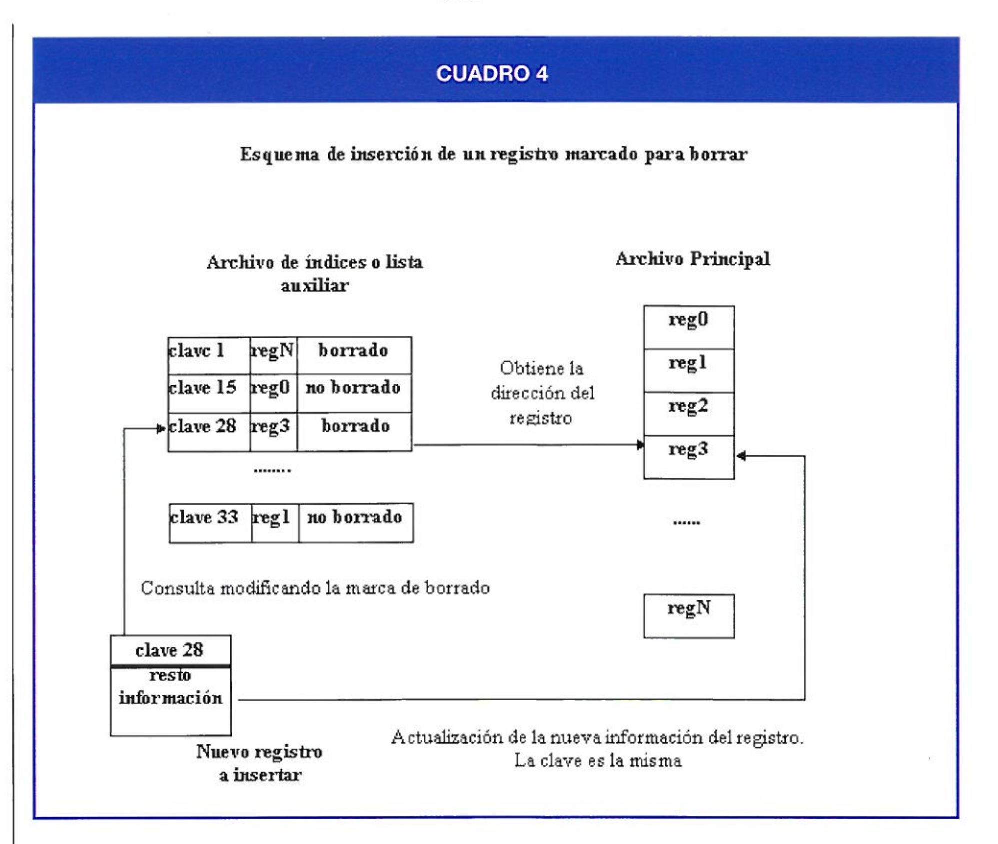
En la unidad listas se pueden encontrar procedimientos y funciones, ya conocidos por los lectores habituales de esta sección, sobre la gestión de listas enlazadas y la definición del registro asociado al archivo de índices, que coincide con la parte de los nodos de la lista correspondiente a la información que almacenan (véase el campo contenido).

La unidad índice contiene la definición del registro del archivo principal, de éste y del archivo de índices. Además contiene todos los procedimientos y funciones para la gestión del archivo de índices. La función *clave_encontrada*, que tiene como parámetros una lista donde se almacenan las entradas de los nuevos registros, el archivo de índices y la clave a buscar, devuelve el valor true si ésta se en-

cuentra en alguno de los anteriores parámetros. Esta función modifica, de encontrar un registro marcado para ser borrado, una variable de tipo lista que apunta al elemento de la lista temporal que contiene dicha clave, para permitir posteriormente la inserción de la nueva información. De forma análoga ocurre con una variable de tipo longint si la clave se corresponde con un registro marcado en el archivo de índices. El procedimiento *fusiona_arch_lista*, que se encuentra en el cuadro 3, acepta como parámetros la lista que contiene las nuevas entradas para el archivo de índices, en el se hace una fusión de ambas estructuras para dar como resultado el nuevo archivo de índices, se apoya en un archivo temporal, del mismo tipo que el archivo de índices, donde se van almacenando en orden los registros. Una vez terminada la fusión se renombra el archivo resultante (previamente se habrá borrado el archivo índice inicial para que sea éste el archivo de índices). El procedimiento *inserta_registro*, cuyos parámetros son el archivo principal, la lista de las nuevas entradas y del nuevo registro a insertar, anexa el registro en la última posición del archivo principal, obteniendo el número de registro donde se insertó éste e insertando en orden un nuevo nodo, que contiene la clave y la posición de inserción. En la lista, esto es cierto para los registros de los que no se ha encontrado su clave dentro de los marcados para borrar. Si la clave introducida se corresponde con un registro marcado para borrar, puesto que en el proceso de búsqueda realizado anteriormente se ha almacenado la posición que ocupa, se procede a utilizar el mismo registro que anteriormente. El cuadro 4 representa esquemáticamente esta idea.

Por último, cabe destacar de esta unidad el procedimiento *comprueba_archivos*. Este procedimiento utiliza la directiva del compilador {$I-} para evitar que al abrir un archivo que no se encuentra en la trayectoria especificada se detenga la ejecución del programa. En lugar de esto se procede a la creación del archivo y posteriormente se procede a abrirlo. Si se desea modificar las trayectorias donde se crearán los archivos se deben modificar las trayectorias asignadas a las variables *tray_arch_prn y tray_arch_ind* en la sección de inicialización de la unidad.

CUADRO 4



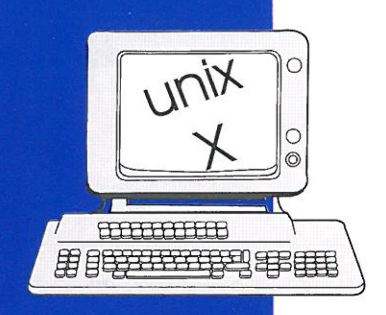
En relación al programa altas, se trata de un programa de demostración muy simple. Comienza pidiendo al usuario si desea introducir más registros. Si la respuesta es afirmativa, se le solicita la que será la clave del registro. Una vez introducida el programa comprueba si ésta existe. Si es así, se visualiza un mensaje indicando al usuario que la palabra ya existe, no permitiendo que se introduzca el nuevo registro. Si no existe ningún registro con dicha clave, se solicita al usuario el resto de los datos, pidiendo finalmente que confirme la grabación del nuevo registro. Este proceso se realiza hasta que el usuario indique que no desea introducir más registros. Por último se procede a la inclusión, en el archivo de índices, de los índices de la lista temporal.

Son numerosas las mejoras que pueden realizarse en el programa: la interfaz con el usuario es muy pobre, se permite introducir un registro cuya clave esté vacía o incluya una cadena de blancos (sería interesante realizar una validación al aceptar un valor en la clave), no se comprueba si el dispositivo donde está almacenado el archivo permite almacenar los nuevos registros, por lo que se podrían perder datos. Tampoco se comprueba si el dispositivo está protegido contra escritura o si los archivos tienen asociados los atributos de sólo lectura u ocultos. En cuanto a la gestión de la lista, si ésta crece demasiado, como la búsqueda en ella se realiza de forma secuencial llegaría un momento que sería bastante ineficiente este proceso, por lo que se podría limitar el número máximo de elementos que puede incluir. Una vez que se llegue a este número se podría realizar la fusión con el archivo de índices, disponiendo a partir de dicho momento de la lista vacía. Se espera que el lector interesado en el tema mejore el programa y las unidades que éste utiliza.

Hay que señalar que falta implementar diversas operaciones que pueden realizarse sobre los archivos que disponen de índices. En el próximo artículo se completarán. Además se describirá cómo se pueden construir archivos de índices correspondientes a claves secundarias y otro método para acceder de forma directa a la información almacenada en un archivo.

CHEQUEO A INTERNET: TCP

Fernando J. Echevarrieta



Este mes continúa en la sección la miniserie dedicada a los protocolos de Internet presentando el protocolo TCP. Un vez más, el tema central servirá como excusa para presentar problemas, soluciones y técnicas empleadas en los protocolos.

Como se recordará de anteriores artículos [1] y [2], la red Internet era una red de conmutación de paquetes basada en el protocolo IP. IP, como protocolo de nivel 3, se encargaba de generar mediante software una red virtual que unía máquinas que no se encontraban directamente conectadas.

Se veía también que en la arquitectura de protocolos por la que se regía Internet, TCP/IP, el último nivel en la torre de protocolos antes de llegar al nivel de aplicación era el nivel 4, de transporte, cuya misión principal era la de facilitar una comunicación directa entre dos puntos finales de comunicación entre entidades remotas [1]. Esta comunicación ya no se realizaba salto a salto, ya que este problema, el de encaminamiento, era labor de IP. Así pues, en este nivel, la red que servía como medio de comunicación quedaba ya totalmente oculta a las aplicaciones.

Por otra parte, en el nivel de transporte, los puntos finales no eran ya máquinas sino que se concretaba más el destino mediante la incorporación de la idea de puerto. Del mismo modo en que cuando se envía una carta a su destino no es suficiente con indicar calle y número, ya que en una misma dirección pueden encontrarse multitud de personas, a la hora de acceder a un servicio no es suficiente con indicar la dirección IP de la máquina que lo aloja, ya que en ella pueden existir multitud de servicios. Así, de la misma forma que en la carta se debe indicar el nombre de la persona, también será necesario indicar el puerto en que atiende un servicio.

Una de las características de IP es que se trataba de un protocolo no fiable, es decir, que no garantizaba el orden en la recepción ni la no existencia de duplicados. Ni siquiera se garantizaba la recepción de las unidades de datos que enviaba, que recibían el nombre de datagramas. Estos problemas se dejaban por resolver al nivel 4. En este nivel, en la arquitectura TCP/IP, existían dos protocolos universales, UDP y TCP. El primero de ellos no aportaba ninguna nueva funcionalidad a IP salvo, claro está, la incorporación de un puerto a las direcciones. Se trataba también de un protocolo no fiable que transmitía mensajes.

Pero la verdadera estrella del nivel de transporte es el protocolo TCP. El objetivo fundamental del mismo es el establecimiento de un servicio fiable orientado a conexión sobre una red de datagramas, lo que lo convierte en el protocolo más complejo de toda la arquitectura TCP/IP, al que sólo es comparable el TP4 de OSI.

FUNCIONES DEL NIVEL DE TRANSPORTE

El nivel de transporte es el primer nivel en el que existe un diálogo directo entre sistemas finales. Por ello, se vuelven a repetir algunas de las funciones que se llevaban a cabo en el nivel 2, de enlace.

Repasando conceptos, esto es natural. Recuérdese que en una red física, el nivel 2 se encargaba de hacer que el medio físico que unía las máquinas fuera semejante al ideal, es decir, trataba, sobre todo, de proporcionar un enlace fiable. El gran invento del nivel 3 consistía en la generación de Internet, una red software virtual que unía redes permitiendo la comunicación entre máquinas que no se encontraban directamente conectadas. Pero ahora, en esta red que ya no es física, será necesario también proporcionar un enlace

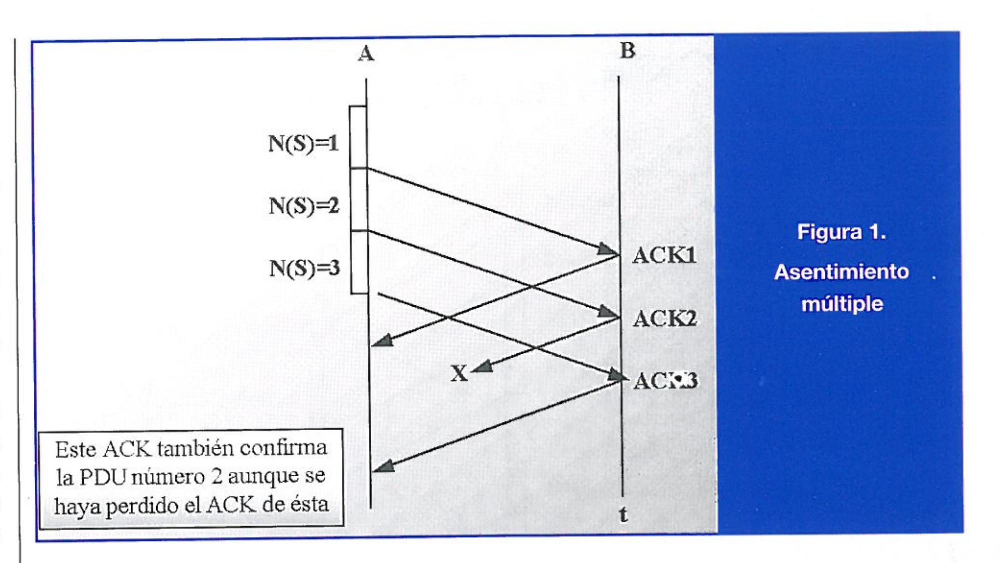


Figura 1. Asentimiento múltiple

fiable entre extremos. Donde antes había dos máquinas unidas por un medio físico que había que hacer fiable, ahora hay dos máquina unidas por un medio lógico, toda una Internet, que también habrá que hacer fiable. Así, por establecer una regla de tres, se podría decir que el nivel 4 es al 3 como el 2 es al 1. Surgen de nuevo los problemas de control de errores, control de flujo, etc.

Pero ahora hay también algunos nuevos, como el establecimiento de la conexión inicial o la memoria de la red. Para el lector que no haya reflexionado sobre ello, piense que Internet es, sin pretenderlo, un gran sistema de memoria. Desde que entra una información por un extremo hasta que sale por otro, esa información se encuentra almacenada en el interior de la red. No de una forma estática, sino circulante, pero, en definitiva, se encuentra en la red. Esta memoria es cuantificable e igual a la suma de los productos de todos los caudales efectivos de entrada de información por el retardo de tránsito de esta información hasta que sale por su destino. Gigas y gigas de información se encuentran permanentemente en tránsito. El problema es que, como veíamos en anteriores artículos, a veces las cosas entran en un orden y salen en otro, o "se mete una y salen tres", problemas que no se trataba de resolver en el nivel 3.

ESTUDIO OSI

Como se indicaba en la introducción, no siempre todos los protocolos de transporte resuelven todos los problemas y proporcionan fiabilidad, como ocurre con UDP. En ocasiones importa más la velocidad que la fiabilidad. Como Vint Cerf, uno de los padres de Internet y el creador de TCP decía, "a nadie le interesa retransmitir un pulso de radar". En efecto, en este caso, es más importante que el siguiente llegue a tiempo que no recuperar uno que se haya perdido. Por otra parte, no todas las redes presentan los mismos problemas. Así, por ejemplo, OSI distingue 3 tipos de redes, las dos primeras orientadas a conexión :

Clase A : Redes totalmente fiables. Servicios perfectos. La probabilidad de reinicialización de la red es despreciable.
Clase B : La entrega de paquetes es perfecta, pero a veces se reinicializa la red.
Clase C : La red no es fiable (en general redes no orientadas a conexión, redes de área extensa basadas en datagramas, radiopaquete, redes internet).

Como es lógico, no será necesario el mismo protocolo de transporte en un tipo de red o en otro. Así, OSI define cinco tipos de protocolo de transporte a los que llama TP (*Transport Protocol*) que se pueden observar en la tabla 1.

Los protocolos TP0 y TP2 se apoyan sobre redes fiables, por lo que son muy sencillos. TP1 y TP3 deben recuperarse de caídas de la red de nivel 3, por lo que deben realizar una nueva conexión, una resincronización y la recolección de pedazos perdidos sin que lo perciba el nivel superior. Pero el más complejo de todos es el TP4, que se apoya sobre una red de datagramas. Las funciones que implementa cada protocolo pueden observarse en la tabla 2. TP4 es el único que se puede comparar al TCP. Así pues, observando las funciones a cumplir por TP4 se observan también las que debe cumplir TCP.

De artículos anteriores [2], se recordará que la segmentación consistía en la división de las *SDUs* procedentes del nivel superior en varios fragmentos y su envío por separado. Esto se hacía generalmente porque el tamaño máximo de las *PDUs* que podía transmitir el nivel inferior era menor que el de la *SDU* que se deseaba transmitir.

Asimismo, los datos acelerados o datos fuera de banda eran aquellos con prioridad sobre el resto y que no esperaban en cola a que llegara su turno para ser entregados. Un ejemplo de estos datos era la pulsación de <CONTROL+C> en una emulación de terminal o una transmisión de un fichero.

La división de conexiones consiste en el uso de distintas conexiones de red para atender a una única de transporte.

Como se puede apreciar, son muchas las funciones a realizar por el protocolo TCP. Pero, pese a que su *PDU* es bastante sencilla, conviene aún exponer algunos conceptos generales sobre protocolos para poder entender sus campos.

TÉCNICAS DE CONTROL DE ERRORES

En arquitecturas de protocolos existen dos formas de afrontar los errores que puedan surgir en un nivel inferior: la corrección de errores en destino o téc-

TABLA 1. Protocolos de transporte OSI

TP	RED	Nombre
0	A	Clase sencilla
1	B	Clase básica de recuperación de errores
2	A	Clase de multiplexación
3	B	Clase de multiplexación y recuperación de errores
4	C	Clase de recuperación y detección de errores

"Funciones de los protocolos de transporte OSI. X : Implementa ; - : No implementa ; O : Opcional"

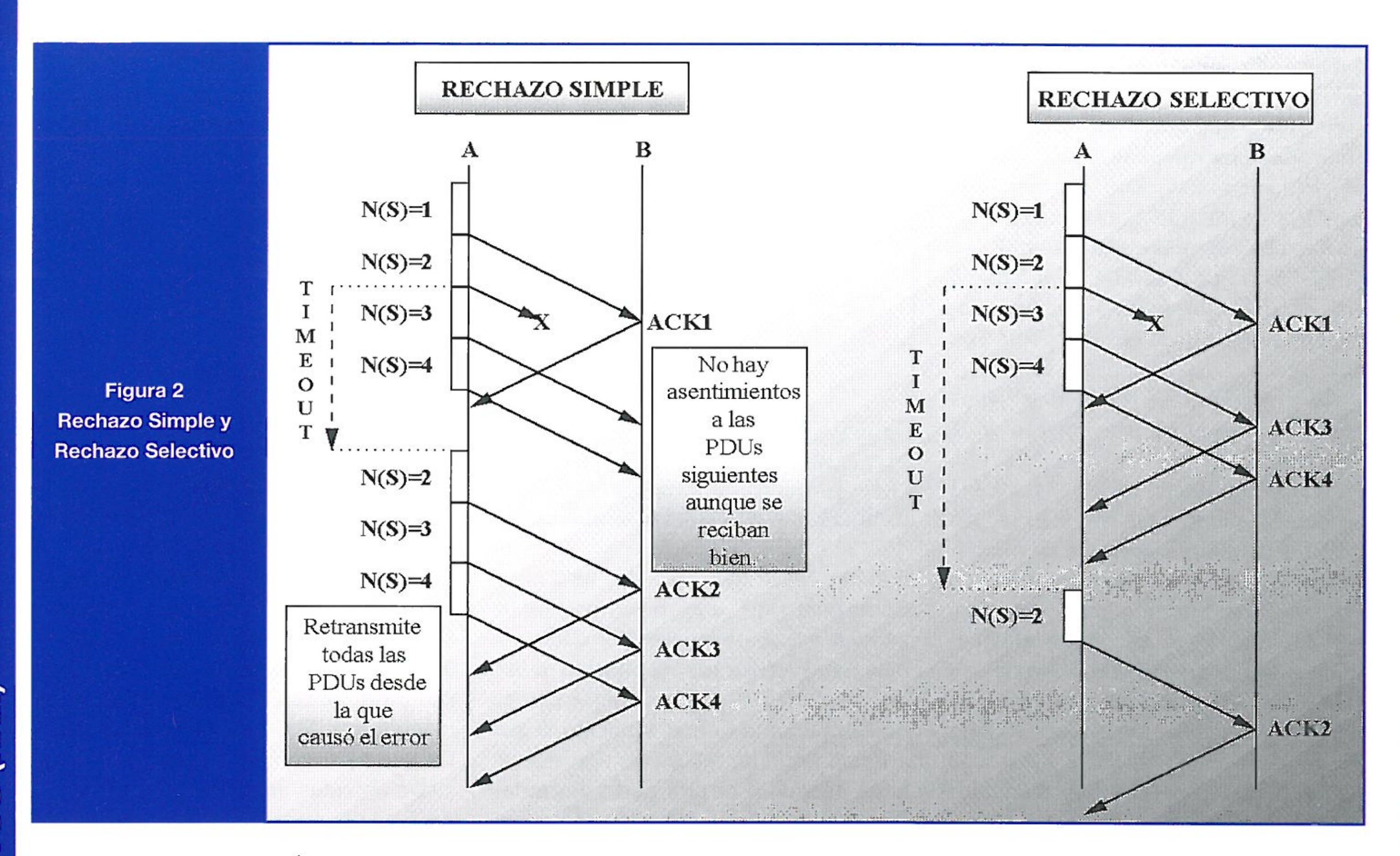


Figura 2
Rechazo Simple y Rechazo Selectivo

nicas FEC (*Forward Error Correction*) y la detección de errores y solicitud de retransmisión o técnicas ARQ (*Automatic Repeat reQuest*).

Las técnicas FEC se basan en la transmisión de gran cantidad de redundancia junto a los mensajes y su principal ventaja es que no requieren un canal de retorno para informar al emisor del estado de la recepción.

Las técnicas ARQ, en cambio, se basan en utilizar la redundancia para detectar errores, pero son mucho más simples ya que no tratan de corregirlos. En lugar de ello, se basan en la transmisión al receptor de mensajes de asentimiento o ACKs (*ACKnowledgement*), confirmando la correcta recepción de los datos que envió. Si el receptor no recibe un ACK transcurrido un tiempo, retransmite automáticamente el mensaje considerando que éste se ha perdido o que fue recibido de manera incorrecta. En algunos casos, se definen mensajes de asentimiento negativo o NACKs que indican que en el mensaje recibido se detectó un error.

El empleo de estas técnicas lleva asociado el problema de que los mensajes de ACK pueden perderse, lo que puede dar lugar duplicados. Imagínese, por ejemplo, que una máquina A envía un mensaje a B, tras lo que activa un temporizador. B recibe el mensaje y confirma la recepción mediante un ACK, pero este ACK se pierde en el camino. En A vencerá el *time-out* del temporizador, por lo que se considerará que el primer mensaje no llegó a destino y se retransmitirá. Si este mensaje no se pierde, en destino se habrá recibido un mensaje duplicado.

BITS ALTERNANTES Y PROTOCOLOS DE PARADA Y ESPERA

Con el fin de protegerse del problema de los duplicados, los protocolos suelen asociar un número de secuencia a sus *PDUs*. De esta forma, si se reciben dos PDUs con el mismo número de secuencia, se sabrá que se trata de un duplicado.

La forma más elemental de emplear números de secuencia es mediante protocolos de bit alternante. Estos protocolos utilizan un sólo bit como número de secuencia, por lo que habrá *PDUs* numeradas como 0 ó 1 de forma alternante. Este tipo de protocolo se denomina de parada y espera, ya que no se permite enviar más de una PDU distinta sin recibir confirmación. Así, si A envía la *PDU-1* debe parar y esperar hasta recibir la confirmación ACK y, si no llega, re-enviará la *PDU-1*. Si B recibe dos *PDU-1* sabrá que se trata de un duplicado.

Algún lector podría pensar que no es necesario realizar una parada con cada envío ya que, al menos, se podrían enviar dos *PDUs*, 0 y 1, sin posibilidad de confusión. Esto es incorrecto, ya que si en destino se recibiera, por ejemplo *PDU-1*, *PDU-0*, *PDU-1* no habría forma de saber si esta última *PDU* es una retransmisión de la primera o una nueva *PDU*. Si B hubiera asentido, ACK-1 pensaría que la tercera *PDU* es nueva, pero si el ACK se hubiera perdido, lo que nunca puede saber B, la tercera *PDU* podría ser una retransmisión de la primera. Si B sabe que no se le permite a A enviar una *PDU* numerada como 0 hasta que esté segura de que la numerada como 1 ha llegado, es decir, que se le obliga a hacer parada, sabrá con certeza que dos *PDUs* consecutivas con el mismo número son retransmisiones y dos *PDUs* no consecutivas con el mismo número seguro que no son retransmi-

TABLA 2. Funciones de los protocolos de transporte

	0	1	2	3	4
Establecimiento conexión	X	X	X	X	X
Segmentación	X	X	X	X	X
Recuperación de caidas	-	X	-	X	X
Concatenación de PDUs	-	X	X	X	X
Datos acelerados	-	O	O	X	X
Control de flujo a nivel 4	-	-	O	X	X
Multiplexación	-	-	X	X	X
División de conexiones	-	-	-	-	X
Reordenación de datos	-	-	-	-	X
Checksum	-	-	-	-	O

siones. De esta forma, la propia numeración de la secuencia sirve al destino como confirmación de la recepción de los ACKs.

PROTOCOLOS CON VENTANA

Un protocolo de parada y espera supone una baja eficiencia, ya que obliga a esperar la recepción del ACK de cada *PDU* para enviar la siguiente. Y este tiempo de espera supone el tiempo de ida, proceso en destino y vuelta, lo que para redes de largas distancias y retardos como Internet es muy poco eficiente.

Para afrontar este problema, surge la idea de permitir que la fuente envíe cierto número de *PDUs* sin esperar confirmación para las mismas. Este tipo de protocolos se denomina protocolos con ventana, y ventana es el nombre que recibe el número de *PDUs* que se permite transmitir sin recibir ACKs. Si se permite una ventana suficientemente amplia como para que vayan llegando los ACKs de las primeras PDUs enviadas, se podrá conseguir un envío continuo. Por otra parte, el propio carácter limitador del uso de ventanas es una solución al control de flujo. Si se permite un tamaño de ventana variable controlado por el receptor, éste podrá disminuir el tamaño de la misma cuando se encuentra saturado, limitando así la velocidad de emisión de la fuente.

La acotación de las ventanas es uno de los múltiples detalles con que se presenta el diseño de un protocolo, ya que de no dimensionarse bien se puede nuevamente caer en el error que se describía en el punto anterior. En algunos casos, como el citado, la solución consiste en emplear, al menos, un número de secuencia más que el de *PDUs* que se permita transmitir sin recibir confirmación. Por ejemplo, si se emplean números de secuencia de 3 bits (8 posibles números: 0..7), la ventana de tamaño máximo será 7, es decir, inicialmente será posible enviar *PDUs* numeradas de 0 a 6. Pero el tamaño de la ventana máxima en relación con los números de secuencia presenta otros problemas. Así, si las *PDUs* pueden llegar desordenadas este tamaño máximo también podrá producir errores.

En cualquier caso, basta entender el concepto de ventana para comprender posteriormente el significado del campo correspondiente en el segmento TCP.

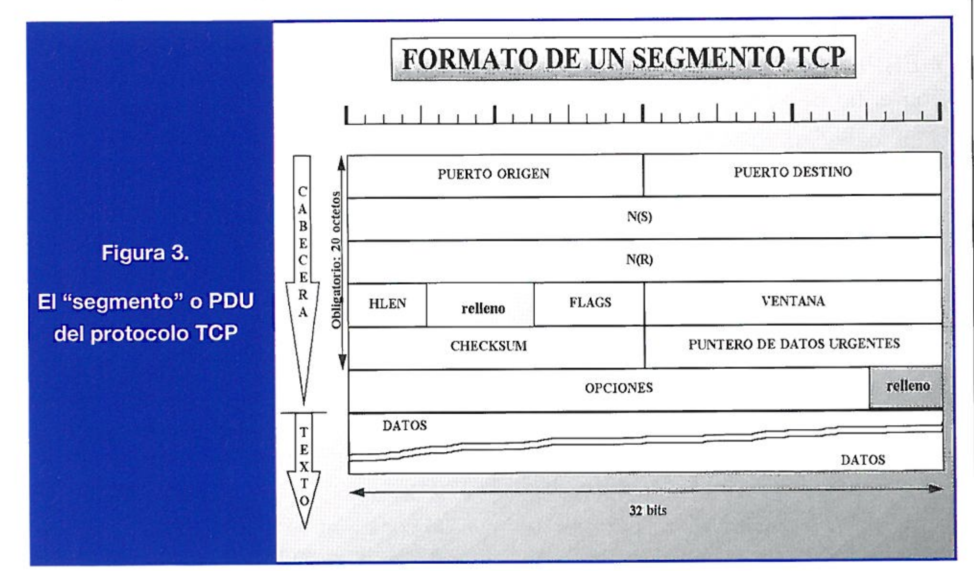


Figura 3. El "segmento" o PDU del protocolo TCP

ASENTIMIENTOS Y RECHAZOS

Como se ha comentado anteriormente, existen dos tipos de asentimientos: positivo (ACK) para confirmar la recepción de una *PDU* en la que no se ha encontrado error, y negativo (NACK) para indicar que se ha encontrado error en una *PDU*. Pero diferentes protocolos asocian también diferentes semánticas a los asentimientos. Así, es posible emplear asentimiento simple, en el que un ACK confirma sólo una *PDU* o asentimiento múltiple, en el que un ACK confirma una PDU y todas las anteriores. En la figura 1 se ha indicado un ejemplo de esto.

Esta figura es un diagrama temporal típico en telemática, en el que se observan dos extremos de una red que separan dos máquinas, en este caso A y B. A cada extremo se asocia una línea y cada punto de ésta es un instante en el tiempo, es decir, el tiempo se representa en vertical hacia abajo. Cada rectángulo representa una *PDU* y su longitud es el tiempo que ésta tarda en transmitirse a la red. Las flechas indican desplazamiento de A a B y viceversa y el tiempo de tránsito a través de la red. Todas tienen sentido descendente, ya que llegan más tarde de lo que salieron. De esta forma pueden medirse los tiempos en vertical. N(S) son los números de secuencia de las *PDUs*. Cuando se pierde una *PDU* se representa por una flecha interrumpida. Aunque los ACKs también deberían representarse como rectángulos, se ha simplificado la imagen.

Del mismo modo, la forma de reaccionar ante los asentimientos define dos tipos de protocolos :

Por una parte se encuentran los protocolos de rechazo simple. En estos protocolos no se producen asentimientos a las *PDUs* siguientes a un error aunque se reciban bien. De esta forma, el origen, una vez vencido el *time-out* de espera de asentimiento de la *PDU* que falló, retransmitirá ésta y todas las siguientes (figura 2).

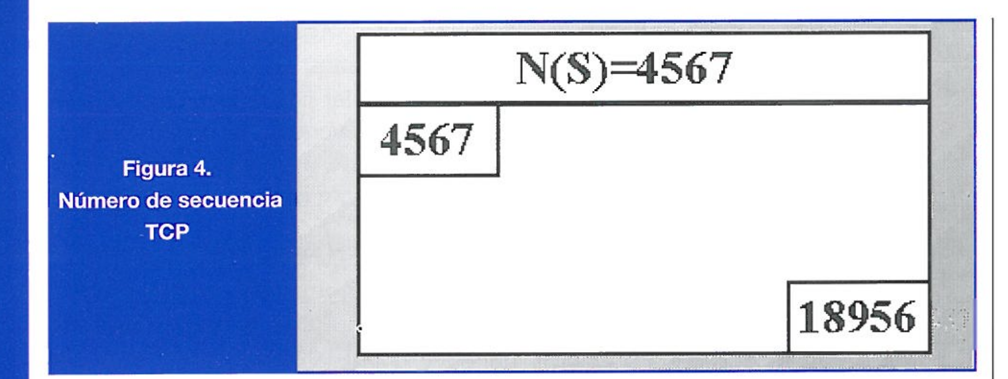


Figura 4. Número de secuencia TCP

En el otro extremo se encuentran los protocolos de rechazo selectivo, en los que se asienten y retransmiten *PDUs* independientes (figura 2).

LA PDU DEL PROTOCOLO TCP

Una vez expuestos todos estos conceptos, la explicación de la PDU del protocolo TCP es inmediata. Esta PDU recibe el nombre de segmento y su estructura, que se puede observar en la figura 3, consta de los siguientes campos.

- **Puertos**
  Un segmento TCP comienza con las direcciones de los puertos origen y destino. Cada pareja de puertos origen y destino identifica una conexión, por lo que no puede haber más de una conexión entre la misma pareja de puertos.

- **N(S)**
  N(S) es el número de secuencia TCP. TCP no numera las PDU's de cada conexión, sino los bytes que transmite. N(S) señala al primer octeto transportado. En recepción se emplea asentimiento múltiple, es decir, el asentimiento de un bloque implica el de todos los anteriores. De esta forma, si se recibe un asentimiento para n, se tiene la seguridad de que se ha recibido desde 0 hasta n. En la figura 4 se puede ver un ejemplo de esto. El conjunto de octetos enviados en el segmento del ejemplo comienza por el que lleva número de secuencia 4567 y termina por el numerado como 18956. Por tanto, el N(S) del segmento será 4567 y el del próximo segmento deberá ser 18957.

  TCP emplea una tecnica ARQ y por cada segmento enviado activa un temporizador. El protocolo no define si se debe emplear rechazo simple o rechazo selectivo, por lo que esto se deja a libre elección de cada implementación, pero obliga a que el destinatario guarde lo que recibe.

- **Checksum**
  Como suele ser usual, información redundante para la detección de errores.

- **Puntero de datos urgentes**
  El puntero de datos urgentes señala un octeto del campo de datos relativo al número de secuencia N(S). Este octeto es el último octeto del mensaje de datos urgentes. La delimitación de este tipo de datos, como la de cualquier otro mensaje, se debe realizar a nivel superior. Se volverá sobre ello en el apartado dedicado a la concatenación de *PDUs*.

- **Hlen**
  La cabecera de un segmento TCP contiene una serie de campos fijos con una longitud de 20 octetos y una parte variable. Por ello es necesario este campo, que indica la longitud de la cabecera, con el fin de poder delimitarla respecto a los datos.

- **Ventana**
  La ventana TCP es una ventana de tamaño variable controlada por el receptor lo que, como se adelantaba al hablar de ventanas, permite un control de flujo en el nivel de transporte. Este tamaño de ventana no puede reducirse más de lo que aún queda por recibir.

N(R) Y EL CAMPO DE FLAGS

La estructura del campo de *FLAGS* TCP se ha detallado en la figura 5. En la misma se puede observar que existen 6 *flags* que se describen a continuación:

- **URG**: Cuando se encuentra a 1 indica que el campo Puntero de datos urgentes se encuentra activo. En caso contrario, puede ser ignorado.

- **ACK**: Cuando se encuentra a 1 indica que el segmento sirve como asentimiento de otro segmento cuyo número de secuencia se indica en N(R). Como se ha expuesto, TCP emplea asentimiento múltiple y se ha elegido que N(R) indique no el último segmento asentido, sino el siguiente segmento que se espera recibir. Así, por ejemplo, un ACK con N(R)=3456 asiente el segmento 3455 y todos los anteriores.

- **TCP** incorpora la técnica del *piggybacking*, que consiste en emplear un segmento de datos para transportar también un ACK. De esta forma, cuando una entidad TCP desea asentir un segmento espera a que haya datos del nivel superior para enviarlos junto con el asentimiento y, así, economizar uso de la red. Sin embargo no se puede abusar de esta técnica, ya que no se debe permitir que en origen salte un *time-out* de espera y se produzca una retransmisión, con lo que se habría tirado por tierra la economía buscada. Por ello, se suele también activar un *time-out* en recepción a la espera de los datos del nivel superior. En caso de que estos no lleguen, se enviará de todos modos un segmento de control con el ACK.
- **SYN**: Es el *flag* de establecimiento de la conexión. Mediante la combinación de los flags SYN y ACK se forman los

Figura 5. Campo de FLAGS de un segmento TCP

URG	ACK	PSH	RST	SYN	END

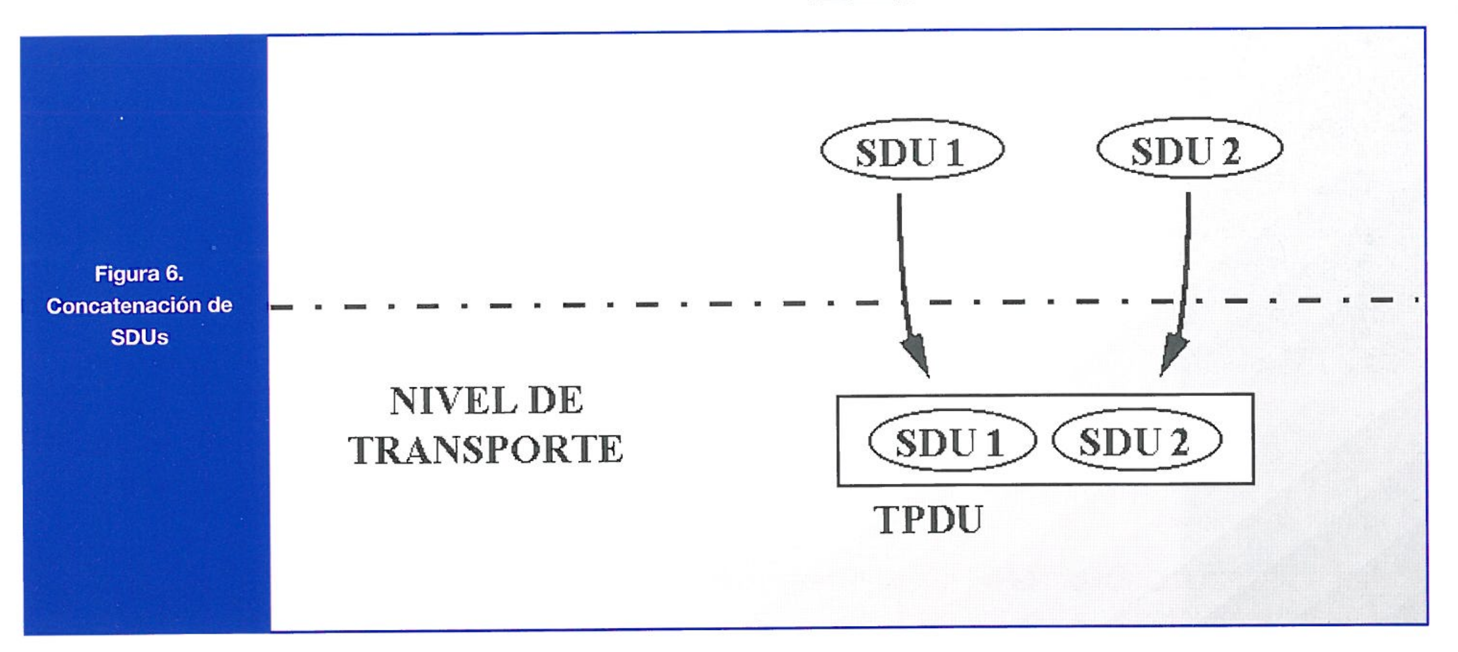


Figura 6. Concatenación de SDUs

mensajes *Connection.REQUEST (SYN=1,ACK=0)* y *Connection.CONFIRM (SYN=1,ACK=1)* del protocolo.

- **RST:** Si se encuentra a 1 indica una inmediata y abrupta liberación de la conexión. Las causas de esto pueden ser la caída de un *host* o que se han recibido mensajes SYN duplicados o que exceden el *time-out*. Como se verá más adelante, la liberación usual de TCP es ordenada.

- **END:** Indica la liberación ordenada de la conexión.

- **PSH:** Solicita una "entrega inmediata" de los datos al usuario del nivel superior. Cuando una entidad TCP recibe un segmento con el *flag* PSH (*PUSH*) activo, "empuja" todos los datos que tenga acumulados al nivel superior.

CONCATENACIÓN DE SDUS

TCP acepta cualquier tamaño de *SDU*. Si es necesario, las divide en fragmentos de 64 Ks y las transmite como datagramas. TCP transmite un flujo de octetos, por lo que no garantiza la delimitación de las *PDUs* de nivel superior. El protocolo únicamente garantiza que el flujo de octetos que se recibe en destino será exactamente idéntico al flujo de octetos que se emite en origen.

Esto puede haber sido comprobado por aquellos lectores que siguieron los artículos de la sección dedicados a programación con *sockets*. Cuando se emplea un *socket* TCP, el *socket* actúa como abstracción de uno de estos puntos de entrada/salida de TCP. Como recordarán los lectores, se leía y transmitía de un *socket* sin necesidad de conocer la red que existía debajo y cada *socket* disponía de un número de puerto y se comunicaba con un destino con otro número de puerto. Al programar comunicaciones mediante *sockets* TCP, había que tener en cuenta que era posible escribir un determinado número de bytes en un *socket*:

write(s,buffer,255)

y, al realizar una una operación de lectura en el otro extremo:

n=read(s,buffer,1024)

recibir una cantidad superior de bytes, por ejemplo *n=700*. Lo que habría ocurrido es que al realizar la operación de lectura en destino se habría extraído de una sóla vez lo que se habia enviado en dos o más operaciones de escritura en origen. Si se interpretara esta lectura como un mensaje, podría generar un error. Por ello, si se emplea TCP para transmitir mensajes de un protocolo de nivel superior, es responsabilidad de las aplicaciones usuarias el delimitar estos mensajes, es decir, sus *PDUs*. La alternativa sería emplear UDP, que sí delimita los mensajes, pero en este caso el programador debería encargarse de gestionar todos aquellos aspectos que no gestiona el protocolo por ser no fiable.

Un ejemplo de esta característica puede apreciarse en la figura 6. En ella se ve cómo el usuario del protocolo (la aplicación que hace uso del mismo), le proporciona dos *SDUs* (*Servide Data Units*). TCP, en lugar en lugar de enviar primero una y luego otra encapsulando cada una en una *PDU* de transporte o *TPDU* (segmento), decide, porque en ese momento le parece bien, y en otro a lo mejor no, encapsular ambas, una a continuación de otra en un sólo segmento y enviarlas juntas. También es posible que se produzca la concatenación de una *SDU* del nivel superior con una *PDU* de control de transporte lista para salir. Este concepto es, por tanto, dual al de segmentación que se presentaba en los artículos dedicados a IP.

Esta característica de concatenación de *PDUs* implica que cuando se produce una retransmisión, el segmento que se envía no tiene por qué ser idéntico al original.

PRESENTACIONES, DESPEDIDAS Y APRETONES DE MANOS

El establecimiento de una conexión TCP se realiza mediante una técnica muy comunmente empleada en las negociaciones de protocolos denominada *triple handshake* (apretón de manos triple). Esta técnica consiste en establecer la conexión mediante tres mensajes:

LIBERACIÓN DE UNA CONEXIÓN TCP

A
B
END
N(S)=i
ACK
N(R)=i+1
Este extremo permanece abierto y puede seguir enviando hasta que envíe un segmento con END=1y le sea asentido
Liberación
No puede enviar más, pero debe seguir escuchando y asintiendo
t

Figura 7. Liberación de una conexión TCP

1. Solicitud de conexión (IDA).

2. Confirmación de conexión (VUELTA).

3. Asentimiento de recepción de la confirmación (IDA).

La liberación de la conexión se realiza también de una forma ordenada. Una "nota de color" es que a la hora de cerrar una conexión primero se cierra uno de los extremos y el otro puede seguir enviando. La figura 7 ilustra este procedimiento. La máquina A envía un mensaje de cierre de conexión (*END=1*), la máquina 2 asiente y cuando A recibe el asentimiento considera liberada su parte de la conexión. Esto significa que ya no puede enviar más, pero sí se encuentra obligada a escuchar lo que B envía y asentir. Para que la conexión quede totalmente liberada por ambas partes será necesario que B realice también este proceso enviando un segmento con *END=1*.

CONCLUSIONES

Con la excusa de presentar el protocolo TCP, se han expuesto multitud de nuevos conceptos como el control de errores, técnicas FEQ y ARQ, asentimientos positivos y negativos, simples y múltiples, protocolos de bit alternante, parada y espera, ventanas, rechazo simple, rechazo selectivo, concatenación de *SDUs* o establecimiento de conexiones mediante triple *handshake*. Con estas ideas y las que se trataron al hablar de IP, se ha tratado de proporcionar al lector una visión amplia de un gran número de aspectos a tener en cuenta en la comunicación entre máquinas. Tras ello, el autor espera que quede más clara la necesidad de la descomposición en niveles de una arquitectura de protocolos que, cuando se presenta sin entrar en materia, suele parecer una estupenda forma de complicar las cosas.

De momento, y como ejercicio para el lector se planteará un clásico problema :

Imagínese dos colinas a ambos lados de un valle. En cada una de las colinas hay un ejército y ambos son aliados. En el valle hay un ejército enemigo. Ninguno de los ejércitos aliados podrá derrotar por separado al ejército que se encuentra en el valle, pero si atacan a la vez la victoria es segura. Los ejércitos se comunican mediante mensajes portados por un mensajero que deberá atravesar el valle, corriendo el riesgo de ser capturado. Así pues, se dispone de una "red", el valle, que transmite mensajes que pueden perderse y sólo se debe atacar cuando ambos ejércitos estén seguros de que su aliado va a atacar también. El lector, que se encuentra en el lado de los aliados deberá pensar un protocolo de comunicación que garantice la victoria.

Piense sobre ello, pero no le dé demasiadas vueltas si no encuentra la solución. Una pista: este problema "tiene truco".

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica relacionada con el artículo, se anima al lector a que se ponga en contacto con el autor mediante:

E-mail Internet: echeva@dit.upm.es
E-mail CompuServe: 100646,2456
WWW:
http://highland.dit.upm.es:8000

REFERENCIAS

Las referencias corresponden a otros artículos de la sección según :

[1], "Redes TCP/IP :
Arquitectura y protocolos", núm. 25
[2], "Chequeo a Internet :
el protocolo IP", núm. 26

AÑADIENDO GRÁFICOS A LA DEMO

Pedro Antón Alonso

La cabecera del formato PCX ocupa 128 bytes, pero es algo que puede ser ignorado si, la resolución y el formato de los ficheros empleados en la demo, son conocidos a priori. No obstante puede resultar de gran utilidad leerla, en el caso de, por ejemplo, sprites. En el listado 3 se puede observar una descripción detallada de dicha cabecera.

Destacar la posibilidad de modificar la cabecera para complicar la descompresión de los gráficos incluidos en la demo, algo realmente sencillo de detectar, pero que, por otro lado, evitará que los gráficos sean "retocados" de forma sencilla. Nótese que es una pobre protección ante cualquiera con unos conocimientos medios del tema.

DESCOMPRIMIR UN PCX

El algoritmo de compresión de los ficheros PCX se denomina RLE (Run Length Encoding)

Descomprimir un fichero PCX implica los siguientes pasos:

En primer lugar, se procederá a calcular las dimensiones en pixels de la imagen, aplicando una sencilla resta:

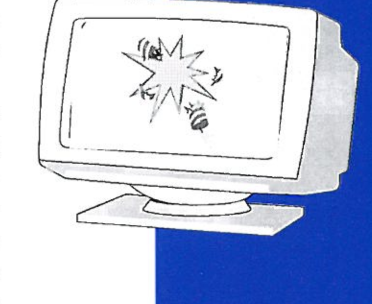
ANCHO = XMax - XMin + 1
ALTO = YMax - YMin + 1

Este paso toma especial relevancia en el caso de que la imagen use más de un plano, para poder calcular el tamaño total de la imagen en bytes. Evidentemente el tamaño de la imagen en bytes, será igual al ancho por el alto por el numero de planos empleado.

Conocida la cantidad de memoria que es necesaria reservar para almacenar la imagen se puede comenzar a descomprimir la imagen. El proceso es realmente sencillo, se lee el primer byte de datos chequeando si éste es mayor o igual de 192, (o lo que es lo mismo, tiene los dos bytes de mayor peso activos) en cuyo caso este byte es el dato a poner en sí mismo. En caso contrario se trata de un contador que indica el número de veces que se debe poner el byte que está inmediatamente después de este byte leído y de valor igual al de los seis bits restantes. Esta operación debe ser realizada hasta completar cada línea horizontal.

En el caso de que la imagen tenga varios planos, los datos que vienen a continuación serán los del siguiente plano de la misma línea horizontal, es por ello que se deben decodificar las imágenes por líneas horizontales.

Este proceso debe ser realizado tantas líneas horizontales como resolución vertical tenga el gráfico en cuestión.



Uno de los formatos gráficos más extendidos es el PCX de ZSoft, y qué duda cabe que una demo necesita incluir gráficos, hasta ahora se había estado trabajando en formato RAW, en este artículo se añadirá a la librería una rutina de carga en formato PCX adaptada a nuestras necesidades

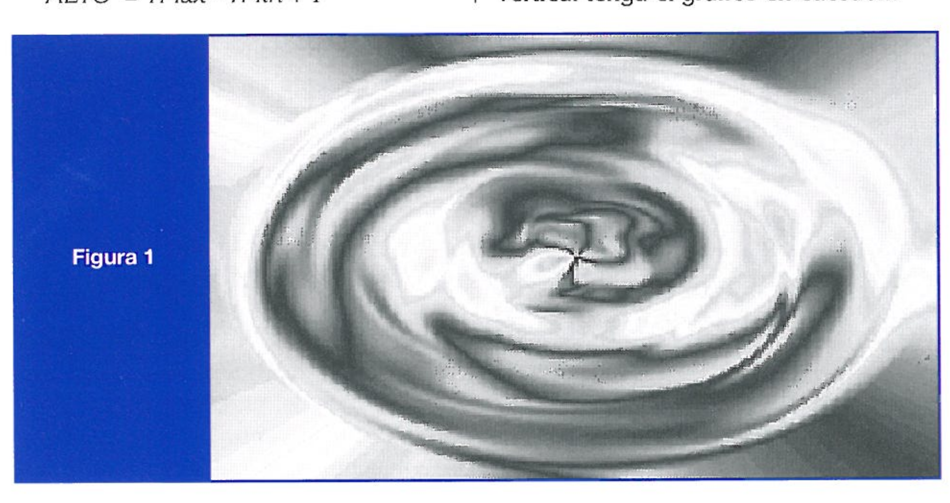


Figura 1

LISTADO 1

```
{ Sólo modos de 256 colores                    }
{ La zona de memoria de descompresión debe tener offset 0 }
PROCEDURE ReadPCXFile
(BufferSeg:Word;NameOfFile:String;OffFile,LenFile:LongInt;PalOnOff:Boolean);
Type PCXHeader = Record
                 Manufacturer : Byte;
                 Version      : Byte;
                 Encoding     : Byte;
                 BitsPerPixel : Byte;
                 XMin, YMin   : Word;
                 XMax, YMax   : Word;
                 HRes         : Word;
                 VRes         : Word;
                 Palette      : Array [0..47] of Byte;
                 Reserved     : Byte;
                 ColourPlanes : Byte;
                 BytesPerLine : Word;
                 PaletteType  : Word;
                 Filler       : Array [0..57] of Byte;
               End;
Var
  PCXFile   : File;
  Header    : PCXHeader;
  Check     : Byte;
  Palette   : Array [1..767] of Byte;
  Width     : Word;
  Height    : Word;
  NumBytes  : Word;
  CntLines  : Word;
  CntBytes  : Word;
  Len       : Byte;
  CntRLE    : Byte;
  BufferOff : Word;

Begin
  If Not FileExists (NameOfFile) then Error ('File not Found',0);
  Assign   (PCXFile,NameOfFile);
  Reset    (PCXFile,1);
  Seek     (PCXFile,OffFile);
  BlockRead (PCXFile,Header, SizeOf (Header));
{ Checking kind of file...      }
  If (Header.Manufacturer=10) and (Header.Version=5) and
     (Header.BitsPerPixel=8)  and (Header.ColourPlanes=1)
     then Begin
          If LenFile<>0 then Seek (PCXFile, LenFile-769)
                     else Seek (PCXFile, FileSize (PCXFile)-769);
          BlockRead (PCXFile, Check, 1);
          If (Check=12) then
            Begin
             BlockRead (PCXFile, Palette,768);
                                 For CntBytes:=0 to 767 do Palette [CntBytes]:=Palette [CntBytes] shr 2;
             Seek     (PCXFile, OffFile+128);
             Width    :=Header.XMax-Header.XMin+1;
             Height   :=Header.YMax-Header.YMin+1;
            { Clip to 320x200 }
             If Width >320 then Width:=320;
             If Height>200 then Height:=200;
             NumBytes :=Header.BytesPerLine;
             If PalOnOff then PutPalette (Seg(Palette),Ofs(Palette));
              for CntLines:=0 to Height-1 do
                Begin
                  BufferOff:=CntLines*320;
                  CntBytes :=0;
                  while (CntBytes<NumBytes) do
                    Begin
                     BlockRead (PCXFile, Check, 1);
                     If ((Check and 192)=192) then
                       Begin
                        Len:=Check and 63;
                        BlockRead (PCXFile, Check, 1);
                        CntBytes:=CntBytes+Len;
                        For CntRLE:=0 to Len-1 do
                          Begin
                           mem[BufferSeg:BufferOff]:=Check;
                           Inc (BufferOff);
                          End;
                       End
                     else
                       Begin
                        CntBytes:=CntBytes+1;
                        mem [BufferSeg:BufferOff]:=Check;
                        Inc (BufferOff);
                       End;
                    End;
                End;
            End
          else
            Begin
             Close (PCXFile);
             Error ('Palette error.',0);
            End;
        End
   else
     Begin
      Close (PCXFile);
      Error ('Not a 256 colour PCX file.',0);
     End;
   Close (PCXFile);
End;
```

LA PALETA

La versión 5 de este formato soporta resoluciones de 256 colores.

Existen varias formas de almacenar la paleta de un fichero en formato PCX, pero se comentarán únicamente dos, la paleta de 16 colores o la de 256.

Para conocer si la imagen posee una paleta de 256 colores, la versión que debe indicar la cabecera es la cinco, y la paleta estará incluida al final del fichero. Si la imagen es de 16 colores la paleta se encuentra en la cabecera.

La paleta está almacenada, como era de suponer, de tres en tres bytes, en formato RGB (Red, Green, Blue) cada byte indica el nivel de rojo, verde y azul que se le indicará al DAC de el subsistema de vídeo. En el caso de tener 16 colores el número de bytes será de 48, y estarán almacenados en orden desde el 0 al 15. Si la imagen posee 256 colores para leer la paleta basta con retroceder 769 bytes desde el final del fichero, en esa posición se encuentra los 768 bytes de RGB de la paleta precedidos de un byte de valor 12. Dividiendo a estos valores por 4 se obtendrá el valor correcto de RGB que se va a enviar al DAC, pero recordemos que en primer lugar se debe verificar que la versión del fichero PCX es la 5.

LA UNIDAD DEMOVGA

El procedimiento ReadPCXFile lee imágenes con formato PCX en 256 colores.

En el listado 1 se puede observar el procedimiento ReadPCXFile, que se encarga de descomprimir un fichero PCX a la zona de memoria deseada.

Esta función no sólo se encarga de leer un fichero PCX, ha sido adaptada a las posibles necesidades de una demo o aplicación gráfica que requiera el manejo de imágenes. Si bien se deben tener en cuenta un par de limitaciones impuestas: el offset de memoria donde se vaya a almacenar la imagen debe ser cero y la imagen debe tener 256 colores.

Pero lo más destacable de la rutina son las posibilidades de manejo:

En primer lugar el fichero, no necesariamente debe estar aislado. Esto quiere decir, que es posible empaquetar los datos del programa y conociendo la posición dentro del fichero donde se encuentra y su longitud, descompri-

mirlo como si de un fichero aislado se tratase.

Y en segundo lugar, la paleta no necesariamente debe ser cargada. Detalle que facilitará los cambios de imagen, y evitará operaciones innecesarias. Evidentemente la paleta de muchas imágenes será común en una demo o proyecto gráfico.

Vista esta introducción damos paso a describir la rutina, que como se puede apreciar en el listado 1, incluye un pequeño kernel, que se ha añadido a la librería, de tratamiento de errores. Una vez leída la cabecera y con un primer vistazo, se puede observar que la función comprueba que el fichero es de tipo PCX, que se trata de la versión 5, que el número de bytes por pixel es 8 y que únicamente se usa un plano de color. Requisitos imprescindibles para que la rutina funcione correctamente.

Acto seguido, se procede a detectar si se trata de un fichero aislado, o si por el contrario, éste se encuentra dentro de un fichero de datos. Para ello la función comprueba si se ha empleado cero como longitud del fichero, en cuyo caso se asume que se trata de un fichero asilado.

A continuación, y situado el puntero de lectura de fichero en la posición adecuada, se lee la paleta, comprobando si el byte leído es 12. Dicha paleta es almacenada en un array local, aunque podría perfectamente almacenarse en la matriz AuxPalette de carácter global, para posibles manipulaciones.

Como el uso de la unidad DEMOVGA está orientado al modo 13h, es decir 320x200 en 256 colores, se calculan los valores ancho y alto de la imagen. Valores que serán almacenados en las variables Width y Height respectivamente y se recortan al ancho y alto de la resolución empleada, es decir 320x200.

Una vez inicializado el contador del numero de pixels por línea, se pone la paleta. Esta última operación será realizada únicamente si la función recibe como entrada de la variable PalOnOff TRUE.

Y por fin, se comienza a descomprimir el fichero, como se ha comentado anteriormente: si el resultado de la operación lógica AND entre el byte leído y el valor 11000000b (192 en decimal) es 192, el dato es un contador, de valor el resultado de la operación lógica AND entre dicho byte y el valor 00111111b (63 en decimal), escribiendo el siguiente byte leído, tantas veces como el contador indique. Si por el contrario el resultado de la operación lógica AND entre el byte leído y el valor 11000000b es distinto de 192, se escribe dicho byte, sin ningún tipo de compresión.

Se insiste en este proceso ya que es el alma de la compresión RLE, y es la única parte de la lectura de un fichero en formato PCX, que puede resultar delicada al lector.

LISTADO 2

```
UNIT DemoVGA;

INTERFACE
{―――――――――――}
{ GLOBAL variables }
{―――――――――――}
CONST
  LogicalWide         = 320;
  LogicalHeight       = 200;
  ClipX1      : Integer =  0;
  ClipY1      : Integer =  0;
  ClipX2      : Integer = 319;
  ClipY2      : Integer = 199;
VAR
  AuxPalette    : Array [0..255,1..3] of Byte;
{―――――――――――}
{ MCGA/VGA functions }
{―――――――――――}
PROCEDURE McgaOn;
PROCEDURE VGA80x50;
PROCEDURE McgaOff;
PROCEDURE VerticalRetrace;
PROCEDURE HorizontalBlank;
PROCEDURE ChangeVGAOff (VGAOff:Word);
PROCEDURE PutColor      (Color,R,G,B:Byte);
PROCEDURE GetColor      (Color:Byte;Var R,G,B:Byte);
PROCEDURE CopyColor     (Desde:Byte;Hasta:Byte);
PROCEDURE RotaPal       (Color1,Color2:Byte);
PROCEDURE PutPalette    (SegPal,OffPal:Word);
PROCEDURE GetPalette    (SegPal,OffPal:Word);
PROCEDURE GetFadePalette;
PROCEDURE FadeUp;
PROCEDURE FadeDown;
{―――――――――――――}
{ MISCELLANEOUS functions }
{―――――――――――――}
FUNCTION  Mcga2Off      (X,Y:Word):Word;
{―――――――――――――}
{ MEMORY functions      }
{―――――――――――――}
PROCEDURE Copy64K       (SegOrg,SegDes:Word);
PROCEDURE Fill64K       (SegDes:Word;Data:Byte);
PROCEDURE CopyBytes     (SegOrg,OffOrg,SegDes,Offdes,Amount:Word);
PROCEDURE FillBytes     (SegDes,OffDes,Amount:Word;Data:Byte);
{―――――――――――}
{ SPRITE functions }
{―――――――――――}
PROCEDURE PutSprite     (SegOrg,OffOrg,SegDes,OffDes,XDim,YDim:Word);
PROCEDURE CopySprite    (SegOrg,OffOrg,SegDes,OffDes,XDim,YDim:Word);
PROCEDURE PutSpriteClip (SegOrg,OffOrg,SegDes:Word;XDes,YDes:Integer;XDim,YDim:Word);
{―――――――――――――}
{ GFX files functions }
{―――――――――――――}
FUNCTION  FileExists    (FileName:String):Boolean;
PROCEDURE ReadRaw       (Var DestinoPtr:Pointer;FileName:String);
PROCEDURE ReadPCXFile
(BufferSeg:Word;NameOfFile:String;OffFile,LenFile:LongInt;PalOnOff:Boolean);
{―――――――――――――――――――}
{ Pix, lines and fillers functions }
{―――――――――――――――――――}
PROCEDURE PutPixel      (SegDes:Word;X,Y:Integer;Color:Byte);
PROCEDURE DrawLineH     (SegDes,X1,X2,Y1:Word; Color:Byte);
PROCEDURE DrawLineV     (SegDes,X1,Y1,Y2:Word; Color:Byte);
PROCEDURE DrawLine      (SegDes:Word;X1,Y1,X2,Y2:Integer; Color:Byte);
PROCEDURE FlatFill      (SegDes:Word;X1,Y1,X2,Y2,X3,Y3:Integer;Col:Byte);
PROCEDURE GouraudFill   (SegDes:Word;X1,Y1,X2,Y2,X3,Y3:Integer;Col1,Col2,Col3:Byte);
```

LISTADO 3. FORMATO DE LA CABECERA DE UN FICHERO PCX por Zsoft

Byte	Contenido	Longitud	Descripcion
0	Manufacturer	1	Constante 10 = ZSoft .PCX
1	Version 0 = Version 2.5 2 = Version 2.8 3 = Version 2.8 5 = Version 3.0	1	Version:
2	Codificación	1	1 = Codificación RLE
3	Bits por pixel	1	Número de bits por pixel de cada plano.
4	Tamaño (Xmin, Ymin) - (Xmax - Ymax)	8	Dimensiones del dibujo en pixels, limites incluidos.
12	HRes	2	Resolucion horizontal del modo de video.
14	VRes	2	Resolucion vertical del modo de video.
16	Paleta	48	Paleta en los modos de 16 colores.
64	Reservado	1	
65	NPlanos	1	Numero de planos de color.
66	Bytes/linea	2	Numero de bytes en cada plano de color por cada linea horizontal.
68	Tipo de paleta 2 = Escala de grises.	2	1 = Color/Blanco y negro
70	Relleno	58	Espacios en blanco hasta completar los 128 bytes de cabecera.

NOTA: Las variables de tamaño 2 son todas enteros.

EL PROGRAMA PCX2RAW

Algunas veces es conveniente almacenar las imágenes en formato puro.

Los ficheros con compresión PCX son muy empleados en el mundo de la demoscene debido a su rapidez de descompresión. Con pequeñas modificaciones de la función comentada la velocidad del proceso de descompresión se puede ver multiplicada por 100. Bastará con leer el fichero completo a memoria, y descomprimir el fichero, una vez que éste se encuentre en la memoria de nuestro ordenador.

Pero el lector estará preguntándose qué ocurrirá si el gráfico almacenado emplea muchos valores superiores a 192, o peor todavía, si los valores son poco repetitivos. La respuesta es obvia, el fichero puede llegar a ocupar más que el tamaño de la imagen descomprimida. En cuyo caso se debe recurrir a otro tipo de compresión, o sencillamente a almacenar la pantalla de vídeo tal cual, de esta última opción se encarga el programa PCX2RAW que también acompaña a este artículo.

Poco se puede decir de este programa salvo que nos permitirá ver la imagen que se va a proceder a almacenar y que es posible almacenar la paleta al final del fichero, en otro fichero, o no almacenarla. Por lo tanto, bastará con indicar al programa el nombre del fichero que se desea convertir, el nombre del fichero donde se va a almacenar el gráfico en formato RAW así como las opciones que se deseen poner.

ÍNDICE DE FUNCIONES DE LA UNIDAD DEMOVGA

La unidad DEMOVGA, que ha sido creada a lo largo de este curso ha ido creciendo, y se ha reestructurado, para un mejor manejo de ella. En el listado 2 se puede observar todas y cada una de las funciones que implementa dicha librería.

La estructuración queda, como puede verse, de la siguiente forma:

- **Variables globales.** Estas variables son manejadas por varias funciones de la unidad. Representan el ancho y alto lógico de la resolución empleada, así como los valores de recorte, y una matriz de 768 bytes donde se puede almacenar una paleta.

- **Funciones MCGA/VGA.** Estas funciones manejan el hardware del subsistema de vídeo que se está empleando y sobre el que se está basando este curso. Con ellas se puede controlar tanto el CRTC como el DAC de un subsistema de vídeo compatible VGA.

- **Funciones diversas.** En este apartado, de momento, sólo se ha incluido una pequeña función que devuelve el offset de vídeo correspondiente a un par de valores (x,y).

- **Bloques de memoria.** Copiar, mover o rellenar zonas de memoria son las funciones que realiza esta parte de la unidad.

- **Funciones de sprites.** Estas funciones permiten poner un sprite dentro de un área, así como copiar un sprite a una zona de memoria.

- **Funciones de ficheros.** En esta parte existen tres funciones, la primera de ellas comprueba si existe un fichero, las otras dos se encargan de leer un fichero RAW y un PCX a una zona de memoria.

- **Funciones de pixel, líneas y rellenados.** Hasta el momento, esta unidad que aumenta con cada número de la revista, permite dibujar pixels, líneas, y realizar rellenados planos y gouraud.

Y para finalizar, desear desde estas líneas una feliz Navidad a todos los lectores y programadores en general.

PROTOCOLOS DE COMUNICACIONES

María Jesús Recio

Un controlador de red de área local (LAN, Local Area Network) es un módulo software que proporciona un interfaz entre una tarjeta de red y el software de comunicaciones de la máquina. Los controladores de tarjetas se diseñan para una tarjeta específica, y habitualmente se proporciona con la tarjeta de red. No obstante, existe un controlador de tarjeta de red "universal" que se puede utilizar cuando no se dispone del controlador propio de la tarjeta y que suele funcionar casi siempre. Se trata del controlador denominado NE2000.

Los controladores de tarjetas se diseñan para una tarjeta específica

Pero no basta con instalar este software en la máquina en la que se ha montado la tarjeta de red, sino que además hay que configurarlo. El programa de configuración de cada tarjeta también se suministra con la tarjeta. En la configuración deben definirse parámetros como son el número de interrupción asignado a la tarjeta (recuérdese que la prioridad de la interrupción es mayor mientras menor sea el número asignado, por lo que es conveniente asignar una interrupción baja (la mínima posible es la 3), para que el funcionamiento de la red sea aceptable) y la dirección de puerto asignada (lo habitual es la 300, si no se tiene ocupada con otro dispositivo). Se debe tener mucho cuidado al realizar la configuración, ya que si se asigna tanto una interrupción como un puerto que ya están siendo utilizados por otros dispositivos, se producen problemas de funcionamiento.

Una vez que la tarjeta está bien configurada y no existen conflictos con otros dispositivos, ya sólo falta instalar el software de red con el que se quiere trabajar.

En la figura 1 se muestra la localización del controlador de la tarjeta de red en relación con el modelo de interconexión de sistemas abiertos. Como se puede observar en la figura, los controladores de red se localizan por en el subnivel más bajo de los dos en los que se divide el nivel de enlace (MAC y

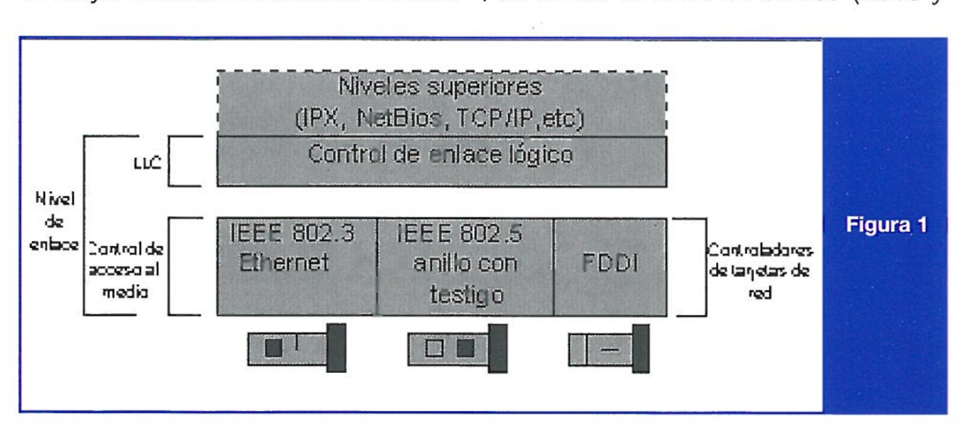


Figura 1

Una red sin protocolos de comunicaciones es como un coche sin combustible. De nada sirve tener grandes equipos, buen cableado, tarjetas de red con DMA. Si no se tienen protocolos de comunicaciones adecuados y bien configurados, no se tiene nada

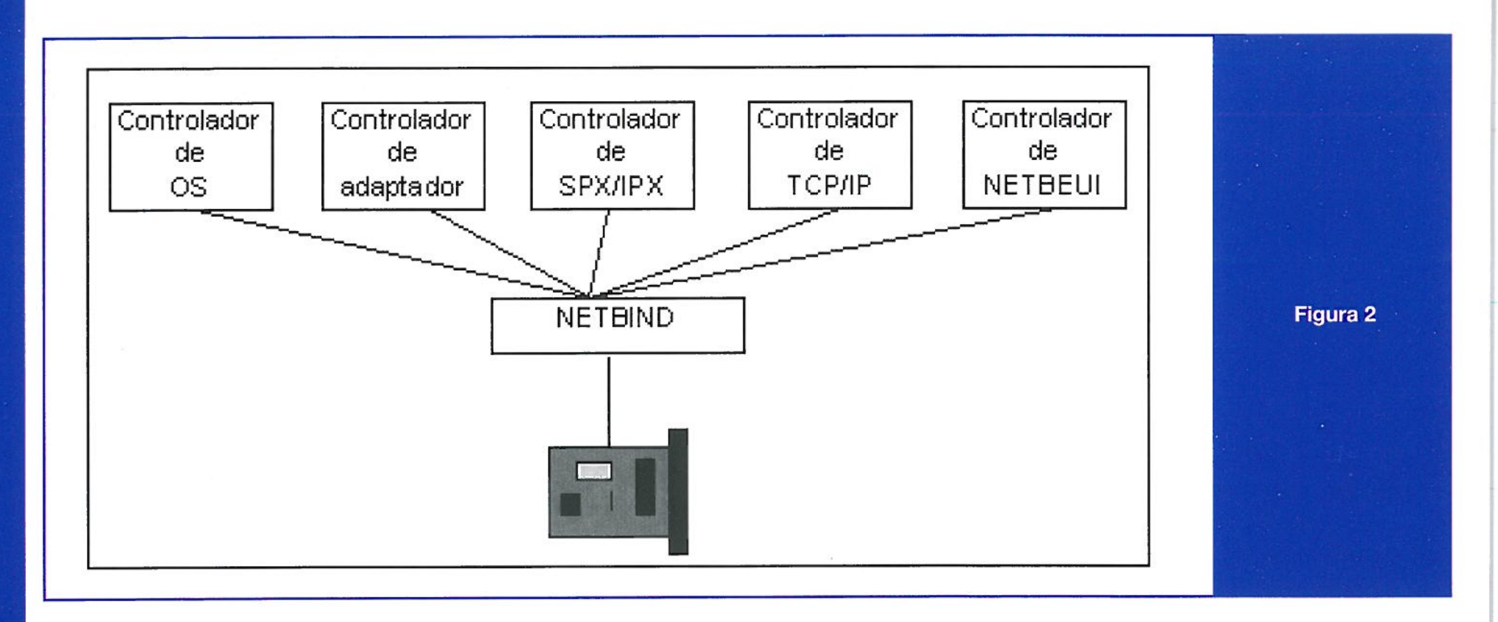


Figura 2

LLC), es decir, se encuentra en el subnivel de control de acceso al medio (MAC) y, por tanto, por debajo del nivel de control de enlace lógico (LLC).

El subnivel LLC es el que realiza las funciones de conmutación de datos entre múltiples controladores de tarjetas de red, caso en el que se dispone de varias tarjetas de red.

Pero no basta únicamente con instalar el controlador de la tarjeta de red. Además de hacer esto, es necesario integrarlo dentro de la pila de protocolos que se vaya a utilizar, es más, se debe integrar en todas y cada una de las pilas instaladas en un ordenador.

El controlador puede instalarse bien de forma independiente, o bien cuando se está instalando la pila de protocolos. Si cuando se instala la pila de protocolos ya está instalado el controlador de la tarjeta, durante el proceso de instalación de la pila se debe indicar (en el caso de no ser detectado por el programa de instalación, que es lo más frecuente) el controlador que se ha instalado para su integración. Si el controlador no se ha instalado previamente, entonces es en este momento cuando se instala e integra a la vez.

Se debe tener cuidado en el momento de seleccionar el controlador de red cuando se está instalando la pila de protocolos, ya que aunque el controlador esté instalado, si no está cargado en memoria no es detectado por el proceso de instalación de la pila de protocolos, y esto puede dar lugar a confusiones.

Todos los ordenadores interconectados deben tener los mismos protocolos de comunicaciones

POSIBLES PROBLEMAS EN LA CONFIGURACIÓN

Además de problemas como los que se han citado (conflictos con la interrupción y puerto asignados) y que son muy importantes, existen otros tipos de problemas que van más allá de la propia configuración de la tarjeta.

Paralelo a NDIS existe otro inferfaz, llamado ODI, desarrollado por la casa comercial Novell

En ocasiones, cuando se intenta conectar varios equipos, a pesar de que todos estén bien configurados, que el cableado se encuentre en buen estado, y que las tarjetas de red funcionen correctamente, los ordenadores no consiguen verse. El problema puede ser tan sencillo como que alguna de las tarjetas de red instaladas en los equipos tenga una dirección física de red que coincida con otra. Aunque esto en teoría es imposible, ya que las direcciones físicas de red no pueden repetirse para garantizar la imposibilidad de ver el problema antes citado, en la práctica no es así.

La dirección física de una tarjeta está dividida en dos partes: una que identifica al fabricante y otra que identifica la tarjeta de red propiamente dicha. Como cada fabricante tiene asignado un número diferente no debería producirse este problema. Sin embargo, existen fabricantes piratas que copian direcciones de red de otros fabricantes. Si se monta una red en la que se produce la casualidad de confluir dos tarjetas de red con la misma dirección física, la red no va a funcionar.

PROTOCOLOS

Los protocolos son el software que permite las comunicaciones no sólo a través de redes de área local, sino también a través de medios como es vía telefónica, vía puertos serie/paralelo del ordenador, etc. En realidad no se

habla de un único protocolo, sino de una pila de protocolos dentro de una arquitectura de niveles. Como ya se mencionó en artículos anteriores de este serie, cada nivel se implementa a través de uno o más protocolos que manejan subsistemas o funciones del proceso de comunicación de forma, por tanto, que cada nivel define una serie de funciones a cubrir que deben ser tratadas en los protocolos elaborados para dicho nivel y utiliza funciones que los protocolos de niveles inferiores deben suministrar.

Una vez que el cableado de la red está listo, es necesaria la puesta en funcionamiento de un software especial, a veces integrado dentro del propio sistema operativo y otras veces no, que permita el claro entendimiento entre las diferentes máquinas. De esta frase se debe sacar una conclusión muy importante: "Todos los ordenadores interconectados de una manera u otra deben tener los mismos protocolos de comunicaciones para posibilitar el entendimiento entre ellos".

Para entender mejor esto, piense un instante en un grupo de personas, cada una de ellas conocedora de un idioma diferente, intentando transmitirse información de unos a otros. Obviamente, esto no es posible. Para poder comunicarse claramente todos deben hablar el mismo idioma.

TABLA 1. PILAS DE PROTOCOLOS MÁS DIFUNDIDAS

PILAS DE PROTOCOLOS
Modelo de interconexión de sistemas abiertos (OSI).
Arquitectura de sistema en red de IBM.
DECnet de DEC.
AppleTalk de Apple.
Internet, inclusive TCP/IP.

Antiguamente sólo se podía trabajar con una pila de protocolos a la vez. Cuando un ordenador quería comunicarse con otro, ambos debían tener cargados los mismos protocolos de red. Si la pila de protocolos cargada en uno de ellos era diferente, entonces se debía descargar esta pila y cargar la pila de protocolos que tenía el otro equipo. Dicho de otra forma, era necesario hacer modificaciones en algunos ficheros de configuración de la máquina, además de otras operaciones, para que surtiera efecto el cambio de pila de protocolos.

NDIS permite que múltiples pilas de protocolos direccionen una misma tarjeta de red

Esto imposibilitaba acciones como tener conexión simultanea a dos máquinas que funcionasen con distintas pilas de protocolos (por ejemplo, un servidor de Novell con IPX y una máquina UNIX con TCP/IP) a través de un entorno multitarea.

En la actualidad, con la especificación del interfaz del controlador de red (NDIS) diseñada para proporcionar a los usuarios acceso a distintos protocolos, no es necesario cargar y descargar protocolos, sino que se permite la convivencia en una misma máquina y de manera simultánea de varias pilas de protocolos diferentes.

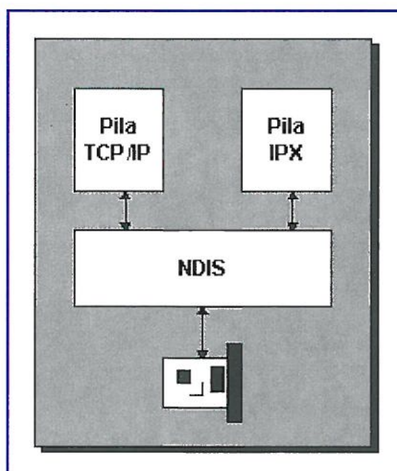
La especificación de la interfaz del controlador de red (Network Driver Interface Specification), como ya se ha mencionado, fue diseñada para proporcionar a los usuarios acceso a distintos protocolos. Se trata, por tanto, de un elemento que hace de intermediario entre los protocolos de comunicaciones y el controlador de la tarjeta de red. De esto se deduce que a la hora de diseñar los protocolos no se tiene por qué saber nada de la tarjeta de red, es decir, no existe un interfaz de tarjeta de red concreto, sino unas especificaciones genéricas que son con las que se trata (ver figura 2).

NDIS permite dos configuraciones distintas:

- Que múltiples pilas de protocolos direccionen una misma tarjeta de red.
- Que cada pila de protocolos gestione una tarjeta de red diferente instalada en la misma máquina. En este último caso es necesario instalar el controlador de cada tarjeta de red por independiente, incluso si son iguales, ya que además el controlador de la tarjeta tiene información acerca del puerto, la interrupción de la tarjeta y, obviamente, cada tarjeta debe tener valores diferentes.

Paralelo a NDIS existe otro inferfaz, llamado ODI (interfaz abierta de enlace de datos) desarrollado por la casa comercial Novell que, al igual que NDIS, permite la coexistencia de varias pilas de protocolos en la misma máquina.

Las ventajas que ofrece el poder tener cargadas varias pilas de protocolos a la vez son obvias: Imagine que necesita conectarse a un servidor de Netware con IPX y a una máquina



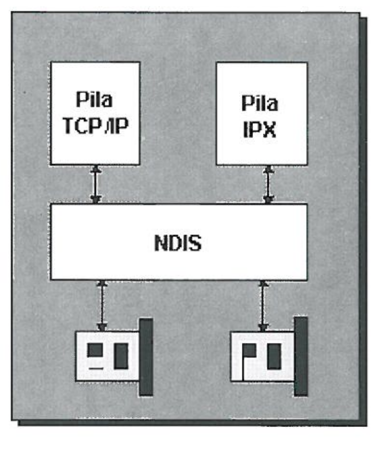


Figura 3

TABLA 2 a. PROTOCOLOS DE LAS PILAS A NIVEL DE APLICACIÓN

PILAS	PROTOCOLOS ESPECÍFICOS A NIVEL DE APLICACIÓN
OSI	FTAM (Acceso y gestión en la transferencia de ficheros). Terminal virtual de OSI. DTP (Procesamiento de transacciones distribuidas). X.400 (Sistemas de gestión de mensajes). X.500 (Servicios de directorio).
IBM	LU 6.2 (APPC Comunicación avanzada programa a programa).
AppleTalk	AppleShare AFP (Protocolo de clasificación de AppleTalk). ASP (Protocolo se sesión). PAP (Protocolo de acceso a la impresora).
Internet	SMTP (Protocolo básico de transferencia de correo). FTP (Protocolo para la transferencia de ficheros). SNMP (Protocolo básico para la gestión de red).

UNIX con TCP/IP. Bastaría que cargase las dos pilas de protocolos y así poder conversar con ambas máquinas de manera simultánea, sin tener que estar reinicializando el ordenador cada vez que se quiera conmutar ente la comunicación de una máquina u otra.

La figura 3 muestra las dos configuraciones posibles con NDIS: cómo todas las pilas de protocolos fluyen hacia la misma tarjeta de red física (parte izquierda de la figura) y cómo cada pila de protocolos direcciona una tarjeta de red diferente (parte derecha de la figura).

Como ya ha debido deducir el lector, no se habla de un protocolo de un nivel en concreto, sino de pilas de protocolos correspondientes a los diferentes niveles. Estas pilas de protocolos, por tanto, no definen únicamente servicios para aplicaciones de usuario (niveles superiores), sino que también definen otro tipo de servicios como es, en la mayoría de los casos, servicios para la resolución de problemas en la red, servicios para el control de los modos de encaminamiento de la información, etc..., siempre dependiente del nivel con el que se esté tratando. Se trata de servicios para la gestión de la red, en general. Las pilas de protocolos más conocidas aparecen reflejadas en la tabla 1.

Para cada una de las pilas mencionadas se crea un protocolo para cada nivel establecido. Las tablas 2.a, 2.b y 2.c muestran protocolos de dichas pilas a distintos niveles, y la tabla 3 muestra una comparativa de niveles y protocolos en diferentes pilas utilizadas por los sistemas operativos más difundidos en redes.

ODI.

ODI significa Interfaz abierta de enlace de datos (Open Data-link Interface). Se trata, como ya se ha dicho, de una estructura implementada por Novell para ofrecer soporte simúltaneo a diferentes protocolos de la red.

Cuando llega un paquete de información por la red a la tarjeta, este inferfaz examina dicho paquete y decide hacia qué pila de protocolos enviarlo. De esta forma, cada pila de protocolos ve únicamente aquellos paquetes que van dirigidos hacia ella.

ODI ofrece soporte para las siguientes pilas de protocolos: TCP/IP, IPX, y AppleTalk. Si se hace necesaria la coexistencia de una pila de protocolos diferente es necesario cargar, además, otro software como es ODINSUP, que posibilita la coexistencia de ODI y de NDIS (que a su vez es el que, de forma habitual, da soporte a otras pilas de protocolos como pueden ser Lan Manager, Lan Server, etc..), haciendo de traductor entre ambos.

TABLA 2b. PROTOCOLOS DE LAS PILAS A NIVEL DE TRANSPORTE

PILAS	PROTOCOLOS ESPECÍFICOS A NIVEL DE TRANSPORTE
OSI	COTS (Servicio de transporte orientado a la conexión). CLTS (Servicios de transporte no orientadas a la conexión).
IBM	APPN (Conexión avanzada par a par).
AppleTalk	RTMP (Protocolo de mantenimiento de la tabla de encaminamiento). AEP (Protocolo de eco). ATP (Protocolo de transacción).
Internet	TCP (Parte del protocolo de control de transmisión).

TABLA 2 c. PROTOCOLOS DE LAS PILAS A NIVEL DE RED

PILAS	PROTOCOLOS ESPECÍFICOS A NIVEL DE RED
OSI	CONS (Servicio de red orientada a la conexión). CLNS (Servicio de red no orientado a la conexión).
IBM	APPN (Conexión avanzada par a par).
AppleTalk	DDP (Protocolo de distribución de datagramas).
Internet	IP (protocolo Internet).

APPLETALK

Se trata de un conjunto de especificaciones que describen cómo conectar ordenadores Macintosh de Apple y otros dispositivos (impresoras, por

TABLA 3. COMPARATIVA DE VARIAS PILAS DE PROTOCOLOS REPRESENTATIVAS DE DIFERENTES SISTEMAS OPERATIVOS DE RES

OSI	NETWARE	UNIX	APPLETALK	LANMANAGER
Aplicación	Protocolo principal de	Sistema de clasificación de red	AppleShare	Bloques de mensajes del servidor
Presentación	Netware	(NFS)	AFP	
Sesión	NetBios	SMTP, FTP, SNMP, Telnet	ASP, ADSP, ZIP, PAP	NetBios
Transporte	SPX	TCP	ATP, NBP, AEP, RTMP	NetBEUI
Red	IPX	IP	DDP	
Enlace	Controladores LAN ODI	Controladores LAN Control de acceso al medio	Controladores LAN LocalTalk, EtheTalk, TokenTalk	Controladores LAN NDIS
Físico	Físico	Físico	Físico	Físico

ejemplo) en una red. Inicialmente AppleTalk no permitía la interconexión con arquitecturas en bus y anillo con paso de testigo, exclusivamente posibilita la conexión con un sistema de cableado denominado LocalTalk. En la actualidad soporta otras arquitecturas como las citadas previamente.

Los dispositivos conectados con estos protocolos se asignan ellos mismos y dinámicamente una dirección de red seleccionada de manera aleatoria de un grupo de direcciones posibles cuando se unen por primera vez a la red. La dirección se selecciona aleatoriamente dentro de un rango de direcciones para ser difundida seguidamente por toda la red y, de esta forma, asegurarse de que ningún otro dispositivo se ha atribuido dicha dirección.

En la tabla 3 se pueden observar los protocolos que forman la pila AppleTalk. Se ve que en la parte inferior del nivel de enlace de datos aparecen protocolos referentes a las tres arquitecturas de red citadas (ethernet, anillo con paso de testigo y LocalTalk).

IPX/SPX.

IPX/SPX (Internetwork Packet Exchange/Sequenced Packet Exchange), es decir, intercambio de paquetes entre redes/intercambio secuencial de paquetes, son dos protocolos de comunicaciones desarrollados por Novell. Sirven de intermediarios fundamentalmente entre el sistema operativo de red Netware y la red física.

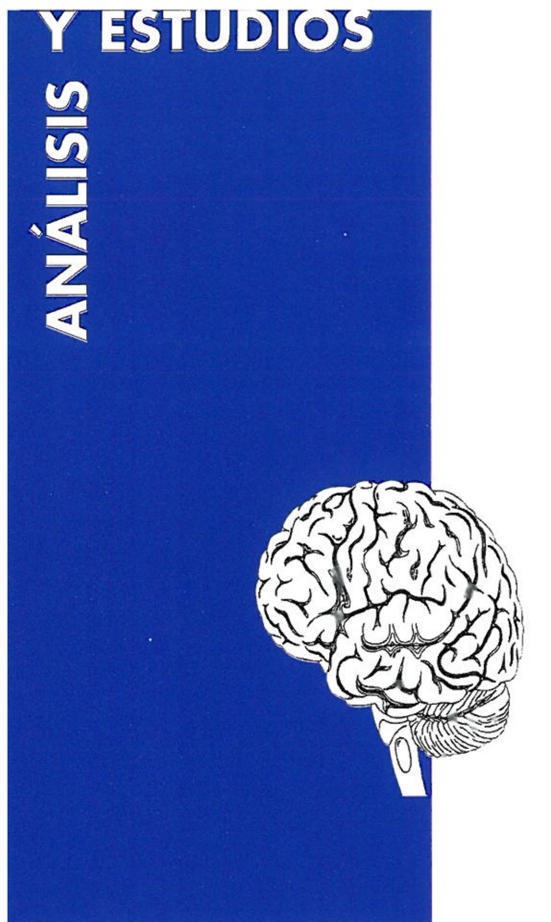
IPX es un protocolo de red, localizado en el nivel de red del modelo OSI. Se trata de un protocolo muy rápido, gracias a características como son la poca sobrecarga de paquetes de control que hace en la red no realiza confirmación de paquetes, se trata de un protocolo no orientado a la conexión, por tanto no implementa la fase de establecimiento y liberación de conexión.

El protocolo SPX se encuentra a nivel de transporte, es decir, por encima del protocolo IPX, ofreciendo mayor complejidad, dando lugar por ello a servicios de mayor calidad como puede ser el tratarse de un protocolo orientado a la conexión, que realiza control de errores, entrega en orden de los paquetes enviados, etc.

Como se puede observar, existe cierto paralelismo entre los protocolos IPX/SPX y los protocolos TCP/IP. No en vano, se trata de protocolos que se encuentran en los mismos niveles (dos a dos) del modelo OSI, por lo que sus funciones deben ser similares.

PRÓXIMO NÚMERO

En el siguiente artículo de esta serie se entrará en profundidad en los protocolos IPX/SPX de Novell, se explicará cómo funcionan, qué se encuentra por debajo y por encima de estos protocolos, y se tocarán otros temas relacionados con dichos protocolos.



ESTADO DE LA INTELIGENCIA ARTIFICIAL

Enrique de Alarcón

La Inteligencia Artificial es el área dentro del mundo de la programación que más importancia tendrá en un futuro próximo. En el presente artículo se va a abordar todo lo relacionado con ella y el estado de evolución en que se encuentra en la actualidad. De cualquier manera es intención de Sólo Programadores dedicar una futura sección a este importante tema.

Cada vez está más cerca el siglo XXI, el siglo que según todos los filmes fantásticos será de un esplendor tecnológico casi indescriptible y en el cual habrá robots inteligentes como principales representantes del mismo. Pero la Inteligencia Artificial, tras cuarenta años de historia desde que en 1956 se celebró la primera conferencia sobre el tema, prácticamente acaba de dar tan sólo su primer paso dentro de su evolución.

Ahora se puede decir que es cuando se empiezan a ver los primeros resultados prácticos reales, con PDA's que reconocen lo que se escribe en su pantalla con bolígrafo, sistemas de dictado que entienden lo que el usuario "habla" mediante micrófono (como el de IBM), robots que reconocen formas y texturas... y que dan una pequeña idea de lo que nos espera en un futuro no muy lejano. Pero como se ha dicho antes, aún pasará mucho tiempo antes de que se puedan hacer realidad robots como los que aparecen en la trilogía de la guerra de las galaxias, por dar un ejemplo familiar.

A continuación se explica todo lo relacionado con la I.A, desde lo que significa por definición, su historia, su evolución, la relación Informática-I.A... y como parte más importante, los principales métodos de resolución de problemas de los que se valen los programadores de I.A. en la actualidad, así como el funcionamiento de las famosas redes neuronales, que parecen haberse convertido en el pilar de todo progreso en este campo.

¿QUÉ ES LA INTELIGENCIA ARTIFICIAL?

Se podría decir que la I.A es un área de la ciencia que estudia la comprensión de la inteligencia para poder crear máquinas inteligentes capaces de realizar igual o mejor las tareas que realizan los humanos (o incluso que estos últimos no son capaces de realizar).

A grandes rasgos se puede decir que la I.A. es resultado del afán humano por mejorar su calidad de vida y que se ha manifestado desde sus propios orígenes.

Actualmente la I.A. se centra en el desarrollo de métodos para la resolución de problemas concretos, para lo cual se basa en dos sistemas diferentes. El primero es el algorítmico y el segundo el heurístico. Los algoritmos son procedimientos de complejidad limitada que dan solución a problemas muy concretos. La heurística, por otro lado, es un conjunto de sistemas que dan solución a problemas de complejidad muy elevada, y de ella se va a hablar más adelante.

Por otro lado, y como se explicará en otros apartados, para problemas de difícil tratamiento digital se usa lo que se llaman redes neuronales (comentadas más adelante).

ORÍGENES HISTÓRICOS

Los orígenes históricos debemos buscarlos en el siglo XVI, cuando se empezaron a construir los primeros autómatas, como son el famoso Pato de Vaucanson, que era capaz de simular la digestión de un pato y mover las alas, o el autómata que simulaba una mano con bastante precisión como para poder tocar correctamente instrumentos como la flauta.

En 1825 Charles Baggage y la Condesa Ada Lovelace investigaron sobre la creación de "calculadores científicos" y "jugadores de ajedrez", basán-

dose para ello en ambos casos en sistemas mecánicos, aunque la realidad es que no llegaron a crear ninguna máquina operativa.

En 1876 Herman Hollerith creó una empresa dedicada a la construcción de maquinas de calculación que se llamaría Tabulating Machine Company, y en 1890 fue capaz de crear una máquina con la cual se podían calcular datos del censo de la población de los Estados Unidos. Después de la guerra, cuando estaba bajo la dirección de Thomas K. Watson, la empresa pasó a llamarse International Business Machinery Corporatión (¡¡¡IBM!!!).

En 1915, Torres Quevedo llegó a construir las dos primeras máquinas del mundo para jugar al ajedrez y pese a que estaban muy limitadas en cuanto a posibilidades de juego, fueron un factor de mucha motivación para otros investigadores.

Todos estos trabajos, tras muchos años de desarrollo, darían paso a lo que hoy llamamos computadores, máquinas capaces de realizar una gran cantidad de cálculos aritmeticológicos a una velocidad impensable hasta entonces.

Una vez aparecidos los computadores, los científicos tuvieron a su disposición una herramienta gracias a la cual se podían programar sistemas capaces de resolver problemas (algoritmos) y a partir de lo cual Church Turing, junto a otros científicos, desarrollaron los conceptos que relacionaban la informática con la formalización del racionamiento humano. A partir de todo esto aparecieron los primeros sistemas matemáticos de razonamiento.

Con todos estos conocimientos presentes ya en la mente de los científicos de todo el mundo, se celebró en 1956 una conferencia en el Darmouth College, situado en los Estados Unidos de América, donde participaron diez de los más importantes científicos de la época en un debate que trato el tema de cómo simular el comportamiento humano mediante sistemas matemáticos y lo cual se podría decir que hizo nacer la I.A. tal como se conoce hoy en día.

1968 fue el año en que nacieron los que se pueden llamar principales grupos de investigación de I.A., que a su vez definieron los principios generales de la misma a partir de toda la experiencia acumulada desde la conferencia de 1956.

En 1962 se publicó un artículo sobre si las máquinas son o no capaces de resolver problemas imposibles para humanos. Pese a que no era un artículo revolucionario, sí lo era el hecho de que había sido escrito en 1947 por un tal Alang Turing, lo cual le convierte en el padre de la I.A.

En 1972 el mismo autor publicó un trabajo llamado Computing Machinery and Intelligence, que consistía en un test o examen que debían superar todas las máquinas antes de poder atribuirles el atributo de inteligentes. Dicho test, por supuesto, no ha sido todavía superado nunca por ninguna máquina creada, y tampoco se espera que ninguna lo haga en los próximos años.

EVOLUCIÓN DE LA I.A.

En los inicios de la I.A. los científicos fueron exageradamente optimistas y predijeron que en poco más de diez años las máquinas ya serían campeonas del mundo de ajedrez y que no mucho después tendrían ya un comportamiento tan inteligente como el humano. La realidad es que el estado actual de la I.A. está muy lejos de las predicciones hechas en 1956, aunque en el campo del ajedrez, por ejemplo, se ha evolucionado mucho y las máquinas están a la altura de los mejores jugadores profesionales del mundo.

Entremos en materia, la evolución de la I.A podemos dividirla en tres etapas. La primera va desde el año 1956 hasta más o menos 1965, la segunda desde 1965 hasta 1975 y la tercera va desde entonces hasta nuestros días. A continuación se detallan los logros más importantes dentro de cada una de las etapas.

● Primera etapa (1956-1965):
En 1956 Arthur Samuel, de IBM, creó el primer programa para jugar a el popular juego de las Damas que además incluía la capacidad de aprender a base de jugar partidas. En 1974, el programa había aprendido ya tanto que fue capaz de ganar a un famoso jugador de damas llamado Doland Michie.

1956 fue el año en que Newell y Simon crearon también el programa LOGIC THEORIST, que fue casi el primero que empleó métodos de resolución heurísticos. Posteriormente, estos mismos autores crearon el programa General Problem Solver, conocido como GPS, que fue un programa destinado a la resolución de cualquier tipo de problema en general mediante el uso, al igual que el anterior, de métodos heurísticos.

En esta etapa se crearon también el primer programa para la comprensión del lenguaje natural escrito y el primer programa traductor, del cual no se consiguieron buenos resultados (actualmente no los hay todavía).

● Segunda etapa (1965-1975):
En esta etapa aparece un nuevo método para la representación del conocimiento (las llamadas redes semánticas), que se basa en que las estructuras de conocimiento se forman a partir de enlazar los llamados nodos y arcos que pueden ser de tres clases: tipo CONCEPTO, tipo EVENTO y tipo CARACTERÍSTICA.

Como programas importantes representantes de esta época aparecen ANALOGY, creado para resolver cuestionarios de inteligencia, QUILLIAN y HENDRIX en el campo de la comprensión de lenguaje natural escrito y el llamado SHRDLU, que fue el más innovador ya que podía discutir sobre cualquier aspecto de las figuras geométricas en lenguaje natural basándose para ello en una técnica llamada de procedimientos, lo cual representó un avance en cuanto a sistemas de comprensión del lenguaje natural se refiere.

En el campo de la robótica, como elemento de mención apareció el SHAKEY, creado por el Stanford Research Institute y que incorporaba un progra-

Los Sistemas Expertos son los máximos representantes actuales de la Inteligencia Artificial

ma de I.A. para generar rutas de desplazamiento basándose para ello en la información visual que recibía (reconocimiento de formas).

También aparecen en esta etapa los primeros Sistemas Expertos como el famoso DENDRAL, que fue operativo a partir del año 1971 y de lo cual se hablará más a fondo a continuación.

● Tercera etapa (1975-Actual):
En la etapa en que nos encontramos la I.A. está evolucionando rápidamente y actualmente los programas "inteligentes" más conocidos son los llamados Sistemas Expertos, que son aplicaciones altamente especializadas en temas científicos concretos que ayudan a profesionales en decisiones importantes y en los cuales se están invirtiendo importantísimas cantidades de dinero para investigación y desarrollo de programas nuevos y más potentes.

Los Sistemas Expertos cada vez son más numerosos y sólo en el año 1984 ya existían unos 138 programas de este tipo dedicados a las más diversas áreas del conocimiento.

En la tabla 1 se detalla una lista de los principales Sistemas Expertos usados a principios de la década de los 80 divididos por áreas de la ciencia. Aparte de los Sistemas Expertos mencionados en esta tabla, actualmente los estudios en materia de I.A. se centran principalmente en los siguientes apartados:

1- Reconocimiento de voz/escritura/imagen.
2- Recuperación inteligente de información.
3- Programación automática (autoprogramación).
4- Robótica en general (que incluye todos los apartados anteriores).

EVOLUCIÓN DE LOS COMPUTADORES

Se puede decir que los computadores son el área de la tecnología que más rápidamente evoluciona, y prueba de ello es que un ordenador en cinco años está totalmente desfasado. Desde que apareció el primer computador hasta nuestros días se pueden diferenciar claramente cuatro generaciones de máquinas que se van a detallar a continuación:

1ª Generación: La primera generación está formada por computadores creados a base de válvulas de emisión termoiónica y se caracterizaban por lo poco fiables e incómodos que eran. Ejemplos de ello son el primer ordenador electrónico, que se construyó en 1946 y al que se puso el nombre de ENIAC. Sus creadores fueron John Mauchly y J.Presper kert. Este computador, para hacernos una idea, pesaba 30 toneladas, ocupaba un área de dieciocho por ocho metros y podía realizar 5000 sumas por segundo. Otro ordenador de esta generación fue el EDVAC, que fue creado en 1947 por John Von Neuman (famoso nombre).

2ª Generación: En ésta se construían ya los computadores a base de transistores, y un ejemplo se puede encontrar en el viejo NCR 304, que se construyó en el año 1957.

3ª Generación: Los pertenecientes a la tercera generación son aquellos que se crearon ya a partir de circuitos integrados de baja y media escala de integración. Ejemplos de estos ordenadores son el IBM 360 y el ICL 1900.

4ª Generación: Estos son los ordenadores actuales, basados en circuiterías de alta integración y que incorporan ya lo que se llama proceso en paralelo. Ejemplos importantes de esta generación son, por ejemplo, los CRAY, el CIBA 205, las estaciones Silicon Graphics, los Digital y todas las gamas inferiores de computadores personales.

LA PRÓXIMA GENERACIÓN

Se ha intentado muchas veces fijar las características que deberán cumplir los ordenadores de quinta generación, pero a grandes rasgos se puede decir que serán las siguientes:

1- Sistema mejorado en el control de la información, ya que los ordenadores de 5ªG se caracterizarán por su capacidad de manipulación de datos.

2- Sustitución de las arquitecturas de circuiterías por otras más rápidas y eficientes para unas mayores transferencias de datos por segundo y aumentar el ancho de banda de los buses.

3- Utilización los microprocesadores en base a otro sistema que sea más rá-

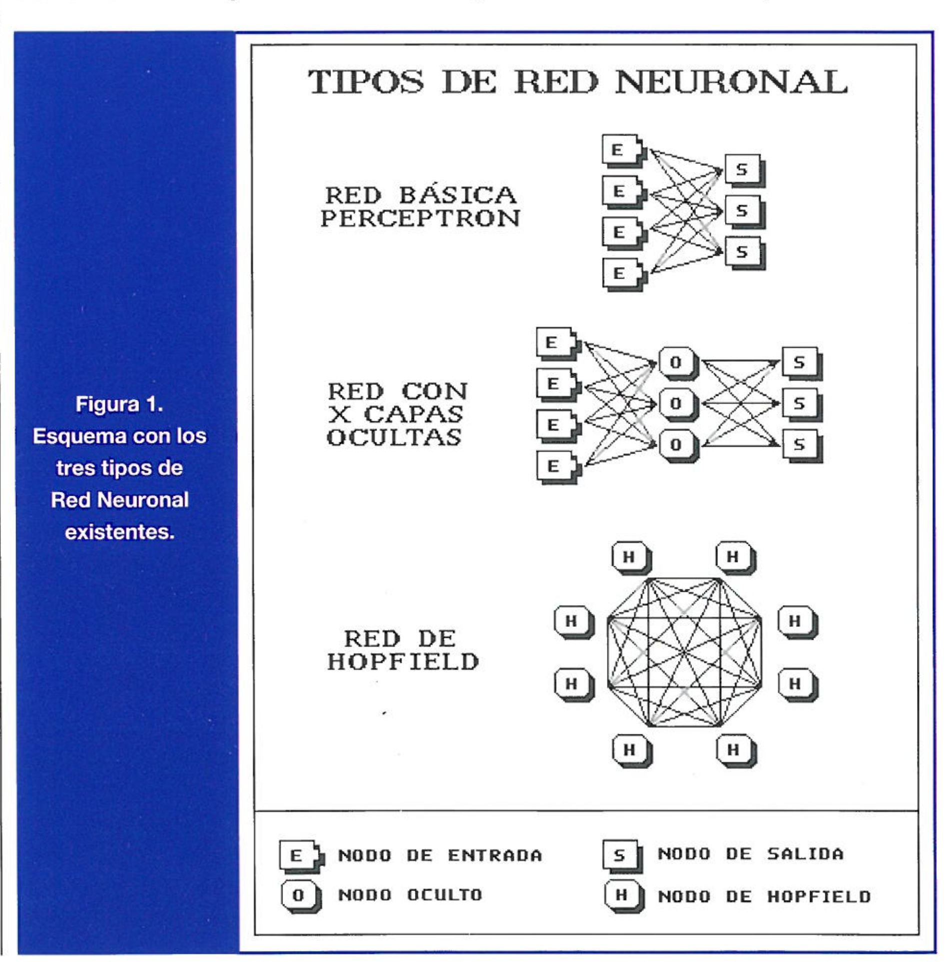


Figura 1. Esquema con los tres tipos de Red Neuronal existentes.

pido a la hora de hacer circular la energía eléctrica a través de ellos que usando transistores como actualmente y/o desarrollar más las investigaciones actuales que hay en microprocesadores de luz, los cuales se basan en la circulación de fotones (luz) en vez de energía eléctrica (electrones),algo que evidentemente hace que sean mucho más rápidos (del orden de entre 100 y 1000 veces más rápidos).

Además, los micros ya no usarán arquitectura tipo Von Neuman, que se basa en la ejecución secuencial.

4- Uso de escalas de integración mucho más elevadas, o sea, más transistores por milímetro cuadrado de superficie de microprocesador.

5- Uso del proceso en paralelo, ya empezado a usar en la cuarta generación. De este modo permitirá la ejecución de un sólo algoritmo de forma compartida entre varios micros.

6- Intentar buscar sustitutos al silicio, que se usa como base de los microprocesadores, que tengan características conductoras mejores que éste.

7- Uso de interfaces inteligentes para la relación con el usuario basándose para ello en la comprensión del lenguaje natural humano (a ser posible oral).

LA PROGRAMACIÓN AUTOMÁTICA

La programación automática es uno de los campos en los que no se ha evolucionado mucho, y el ideal que se persigue es el de crear un programa que a partir de un enunciado concreto (problema) dado por el usuario sea capaz de crear de forma automática un programa para resolver dicho problema. Un ejemplo para entenderlo sería decirle al programa que deseamos un programa (algoritmo) para calcular la posición orbital de los planetas de cualquier momento temporal y que a partir de los datos estructurales del sistema solar (masas, volúmenes , distancias...) él sólo generase el programa capaz de calcularlo.

La realidad es que este proyecto está muy lejos de ser una realidad, aunque los primeros pasos ya se han dado para lograrlo algún día.

Los compiladores convencionales que todo programador conoce son la primera fase de evolución que apareció, y gracias a ellos se hace innecesario programar en código máquina para la creación de algoritmos. El problema es que dichos compiladores son muy rígidos en cuanto a reglas de escritura, y el simple hecho de olvidar un punto y coma puede hacer que el compilador no funcione bien.

Actualmente estamos en la segunda fase, que se caracteriza por la aparición de los primeros programas que programan de forma automática. Pese a estar en fase inicial ya hay varios de ellos, como son el DEDALUS, el PSI y el NLPQ, los cuales dan resultados muy prometedores pese a que no son del todo concluyentes.

I.A. Y ROBÓTICA

La robótica se puede decir que es la unión de todas las áreas de investigación de la informática, ya que para que un ordenador (base del robot) pueda entender el entorno en el que se encuentra debe poder ser capaz de interpretar imágenes, sonido, y otros datos ambientales además de poder controlar todos sus sistemas.

La heurística es lo que se conoce también como lógica difusa

Como ya se sabe, en la actualidad no existe todavía un robot totalmente inteligente, debido principalmente a lo difícil que es que un ordenador interprete el contenido de una imagen o fuente de sonido (ver ejemplo del próximo apartado). Estos dos campos son precisamente los que actualmente están siendo más investigados.

Otro área que se está también investigando mucho es la del procesado de información para la toma de decisiones (que sería la siguiente sensorial obtenida que debería realizar el robot) y para lo cual se están empleando principalmente sistemas de lógica difusa (redes neuronales) de los que se hablará más adelante.

Hablando en términos generales, si para que se considere a un robot como tal debe poder ser capaz de planificar acciones, interaccionar con el entorno y actuar por sí mismo, se puede realizar un esquema de lo que un robot debe poder hacer de forma autónoma para que se le pueda diferenciar de un simple autómata:

1- Debe ser capaz de interpretar su entorno a partir de la información que le den sus sensores (ópticos, acústicos, térmicos...).

2- Poder llevar a cabo los planes de acción que se le den superando las dificultades que se le puedan presentar, como podría ser, por poner ejemplo elemental, esquivar objetos que se interfieran en su posible ruta para llevar una carga de un lugar a otro.

UN EJEMPLO SIMPLE

A continuación se verá un ejemplo muy simple que corresponde a una pequeña parte de la problemática relacionada con la interpretación de imágenes, con lo que el lector podrá hacerse una idea muy rápida de lo complicado que resulta solucionar una cosa tan simple como es la identificación de una textura dentro de una imagen digital, la cual para hacernos la idea podría ser por ejemplo un fotograma digital captado por una supuesta cámara electrónica de un robot (uno sólo entre las decenas que captase por segundo).

Aunque en éste tema concreto no se ha encontrado aún un algoritmo perfecto (¡ni en algo tan simple!) que sea capaz de reconocer cualquier textura en cualquier caso, hay varios ya existentes que dan resultados más o menos aceptables. Los principales son los siguientes:

1- Método de rango: Este sistema se basa en analizar el entorno de cada pixel y dar en cada ocasión como resultado el pixel con un valor lumínico medio entre el pixel de valor mayor y el de valor mínimo de dicho entorno.

2- Método Skewness: Este sistema se basa también en entornos, pero en vez de buscar el valor lumínico medio se basa en aplicar una serie de ecuaciones matemáticas (y que no son sencillas precisamente).

3- Método de comparación: Este es el sistema que tiene para el progra-

mador de a pie más sentido (o que al menos es más fácil de comprender) y consiste en comparar fragmentos de la imagen con un patron predefinido de la textura a buscar. Si la comparación da un valor relativo de igualdad mayor a un 70% (más o menos), la textura se da como buena. Este sistema es sencillo, pero es ineficaz en muchos casos.

4- Método Hurst: Este es un sistema mucho más complejo, ya que necesita muchos más cálculos y usa hasta regresión lineal, pero es también (como pasa siempre) el más efectivo de todos.

Visto esto, se puede hacer una idea de lo que costaría realizar un programa que pudiese, por ejemplo, identificar personas, objetos o lugares dentro de un paisaje o habitación, aunque claro está, esperemos que todo sea cuestión de tiempo.

MÉTODOS HEURÍSTICOS DE LA I.A.

Como se ha explicado al principio del artículo, la I.A. se vale de dos métodos para resolver problemas. El método algorítmico y el heurístico. A continuación se va a hablar de los métodos heurísticos más conocidos, los cuales se conocen también por métodos débiles (el nombre viene dado por su incapacidad para resolver problemas generales) y son los siguientes: Generación y comprobación, búsqueda en amplitud, ascensión de colinas, cumplimiento de reglas, búsqueda por el mejor nodo, reducción de problemas y reducción de diferencias. Dependiendo de qué tipo de problema se plantee, habrá que usar uno u otro método según más convenga.

- Generación/comprobación: Esto consiste en generar una solución posible (un nodo, por decirlo así), comprobar si la solución se sale de los rangos de los resultados posibles aceptables y repetir el proceso en caso de solución no aceptable.

Este sistema funciona bastante bien siempre y cuando el problema no sea muy complicado. Como ejemplo, podemos decir que un sistema parecido a este es usado por el Sistema Experto DENDRAL, comentado anteriormente (Química).

- Búsqueda en amplitud: En este sistema la resolución se basa en ir creando capas (árbol con niveles) de soluciones parciales. Cada vez que se crea una capa con X soluciones posibles se examinan todas y se continúa por la ramificación que parezca más correcta, para ir bajando niveles por el árbol hasta llegar a la solución final. En caso de irse saliendo de los rangos, el procedimiento puede ir retrocediendo dentro de las capas e ir rectificando la solución parcial que va desarrollando.

El único problema de este sistema es la lentitud de resolución que tiene, que aumenta geométricamente a mayor complicación del problema.

- Cumplimiento de reglas: Cuando el problema consiste en descubrir un resultado que respete un conjunto determinado de reglas (o normas), éste es el mejor método que hay.

Este método consiste también en ir generando un árbol con soluciones posibles. Cada vez que realiza un nivel de soluciones, el programa aplica todas las normas que se han dado en el enunciado hasta encontrar un resultado que se ajuste a todos los parámetros.

- Búsqueda del mejor nodo: Este sistema es una combinación de los dos primeros explicados y en algunos casos puede dar buenos resultados en mucho menor tiempo.

- Reducción del problema: Este método se podría decir que se parece al método de búsqueda en amplitud, pero a medida que se realizan niveles de soluciones el árbol se va descomponiendo en pequeños subproblemas que hacen crecer el árbol tanto lateral como descendentemente.

- Reducción de diferencias: Los métodos anteriores se basaban en ir buscando soluciones dentro de un árbol de forma descendente. En este método, las búsquedas de soluciones se hacen simultáneamente hacia arriba y hacia abajo. La parte que busca soluciones de forma descendente partiendo del valor inicial funciona igual que en los

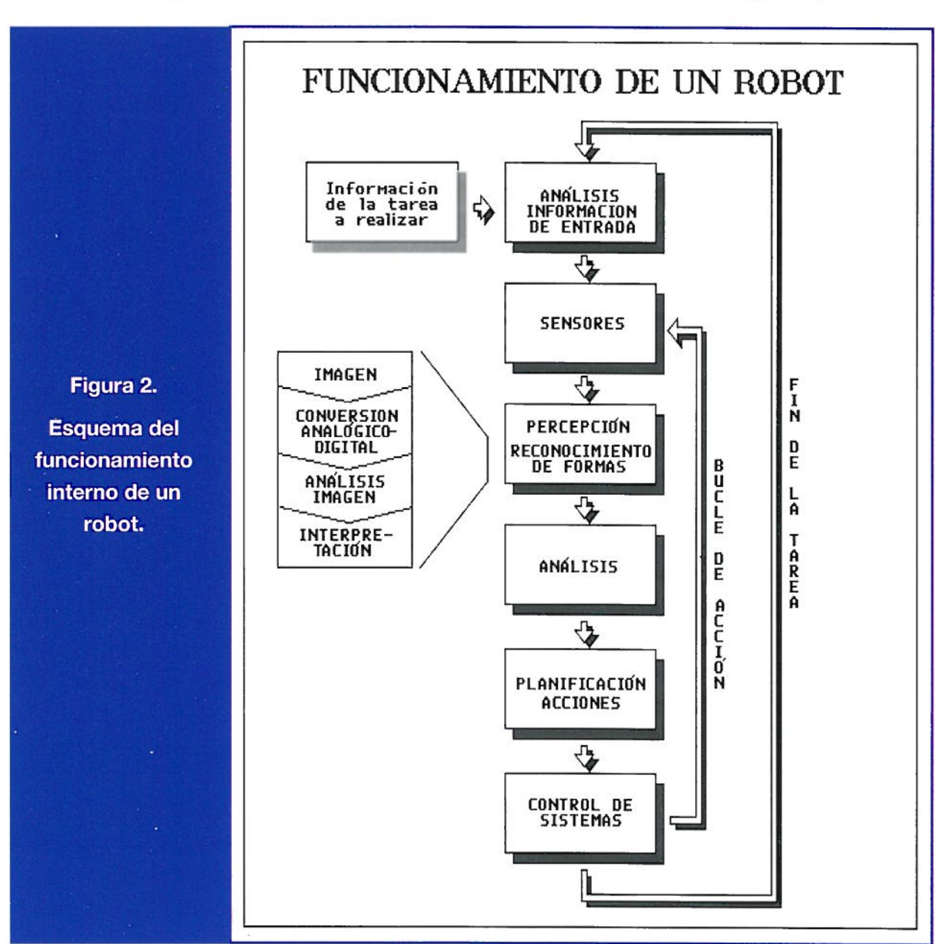


Figura 2. Esquema del funcionamiento interno de un robot.

métodos anteriores, pero además, el sistema crea un valor correspondiente a un nivel bastante avanzado dentro del árbol de soluciones parciales y va buscando soluciones que se vayan acercando al valor de origen (inicio del árbol). En caso de alejarse, se buscan otros valores avanzados de donde partir ascendentemente.

LAS REDES NEURONALES

Las redes neuronales son un conjunto de técnicas que hacen posible la resolución de problemas de difícil tratamiento digital en un ordenador.

Debido a que estos sistemas intentan imitar el funcionamiento del cerebro humano, los sistemas se llaman Redes Neuronales (de neurona). Así, mientras el cerebro tiene millones de neuronas interconectadas entre ellas, las redes neuronales son múltiples nodos interconectados entre sí y que funcionan en paralelo.

El método matemático de imitación de la neurona biológica fue creado en 1943 por W. McCulloc y W.Pitts, de la universidad de Chicago, y su funcionamiento se explica en el siguiente apartado.

Como dato global, se puede decir que las Redes Neuronales están diseñadas especialmente para la resolución de problemas basados en información incompleta o incluso contradictoria, que sería lo que se conoce también por lógica difusa y que proviene de lo que en inglés llaman lógica (fuzzy).

FUNCIONAMIDENTO BÁSICO DE UNA RED NEURONAL

En una Red Neuronal la información a procesar entra por los nodos de entrada. Cada nodo de entrada está conectado a los nodos posteriores (que pueden ser ya los de salida si es una red muy simple) mediante una rama (lo que en una neurona sería un axon) y posee asociado un determinado peso (valor numérico) que le da su importancia relativa dentro de la red.

En las aplicaciones reales los nodos funcionan de modo binario, y por lo tanto sólo tienen dos estados funcionales posibles (0 y 1) o dicho de otro modo, según si el dato de entrada que llega al nodo está por encima o por debajo de un umbral numérico relativo al número de peso del nodo, estará activo o inactivo.

Las Redes Neuronales más sencillas son aquellas formadas por tan sólo dos capas de nodos, los de entrada y los de salida. Este tipo, que se u saba hace años (en los inicios de estas redes) se llaman PERCEPTRON y fueron creadas por Frank Rosenblatt.

En las redes actuales, suelen tener varias capas intermedias entra las de entrada y salida, a estas capas intermedias se les llama hidden layers o niveles ocultos.

Otro tipo de red que se usa también en la actualidad se basa en una estructura que no está estructurada en varias capas superpuestas, sino en sistemas amorfos donde los nodos están todos conectados entre ellos. A este tipo se le llama Red de Hopfield y gracias a este tipo se puede simular otra característica del cerebro humano, que es la de tomar decisiones por fases y no de forma instantánea. Estas redes, en lugar de dar resultados al llegar a la última capa de nodos, funcionan a base de adoptar estados estables de funcionamiento.

CREANDO UNA RED NEURONAL

Cuando se ha programado una Red Neuronal para una tarea determinada, queda por hacer que dicha red aprenda a funcionar con los datos con los que funcionará habitualmente, lo cual consiste en que se debe dar a cada nodo de la red su valor de peso correcto para que después el conjunto de la red de resultados correctos.

Para realizar esto se debe someter a la red a una "clase de aprendizaje", para lo cual hay dos reglas o métodos mediante las cuales llevarla a cabo siempre y cuando se trate de redes de sólo dos capas.

La primera regla es la llamada regla de Widrow-Off y la segunda regla es la del Perceptrón (la más conocida).

La regla del Perceptrón consiste en ir dando varias combinaciones de valores de entrada a la red sabiendo en cada caso el resultado correcto que le corresponde de salida para ajustar así los pesos de los nodos individualmente hasta que coincida el resultado de la red con el que ya se sepa que es correcto.

Las Redes Neuronales, una vez creadas, deben someterse a un aprendizaje

La regla de Widrow-Off es idéntica a la del Perceptrón, pero cada vez que se obtiene un resultado no deseado se reajustan los pesos de todos los nodos proporcionalmente a la diferencia entre respuesta obtenida y deseada.

En el caso de redes de más de dos capas, como se ha dicho, no sirven estas dos reglas y es por ello que se creó en 1974 la regla llamada Método de retropropagación gradiente, que es una versión adaptada de la de Widrow-Off y consiste a grandes rasgos en propagar la diferencia a realizar en los nodos desde la última capa de nodos hasta la primera de forma proporcional a la distancia en capas.

Para realizar todo este proceso de aprendizaje explicado, afortunadamente no hace falta que se haga calculando manualmente, puesto que se tardaría incluso años en acabar, ya que hay casos reales de redes complejas en las que se ha necesitado un superordenador para la fase de aprendizaje debido a la cantidad de información que había por generar y calcular.

Para solucionar esta fase de aprendizaje existen varios programas informáticos que realizan la tarea de forma rápida y sencilla por nosotros. El programa "profesor" más conocido es el llamado BACK-PROPAGATION.

Una vez la red neuronal ha sido programada y enseñada a funcionar, será capaz de resolver problemas que usen valores con los que no se había encontrado anteriormente en la fase de aprendizaje. Actualmente se han conseguido avances muy importantes en el campo de las Redes Neuronales.

Así, por ejemplo, se han conseguido sistemas muy precisos (eficaces) de interpretación de lenguaje escrito a mano, usado por ejemplo en algunos

TABLA 1

MEDICINA:

- MYSCIN: Experto en tratamiento de enfermedades infecciosas.
- CASNET: Experto en la diagnosis y tratamiento del Glaucoma.
- INTERNIST: Dedicado a medicina interna.
- PIP: Especialista en enfermedades Hepáticas.
- PUFF: Dedicado a estudiar las funciones pulmonares.
- HODGKINGS: Experto en el tratamiento de la enfermedad del mismo nombre.
- HEADMED: Experto en todo tipo de psicofármacos.
- VM: Sirve de supervisor para unidades de cuidados intensivos (Hospitales) controlando la evolución y estado de los pacientes.
- ONCOCIN: Supervisor de experimentos en pacientes cancerosos irreversibles.

QUÍMICA:

- DENDRAL: Experto en la identificación de moléculas a partir de espectogramas.
- CONGEN: Se podría decir que es una versión mejorada del DENDRAL que tiene muchas más capacidades analíticas.
- META-DENDRAL: Experto en la creación de reglas a partir de espectogramas de masa basándose en estructuras ya conocidas por el programa. A mayor experiencia adquiere el programa, más exacto se hace.
- CRYSALIS: Experto en el estudio de las estructuras que forman las proteínas.

MATEMÁTICAS:

- MACSYMA: Experto en la resolución de problemas complejos de matemáticas.

GEOLOGÍA:

- PROSPECTOR: Realiza prospecciones de minerales en roca dura. Su eficacia es tan buena que ha llegado a localizar yacimientos de minerales donde los expertos opinaban que no podía haberlos.

EDUCACIÓN:

- SCHOLAR: Experto tutor de geografía.
- SOPHIE: Profesor en la búsqueda de averías en equipos electrónicos.
- EXCHECK: Profesor de lógica y de la teoría de conjuntos.
- HEARSAY: Intérprete del lenguaje natural humano.

Principales Sistemas Expertos usados a principios de la década de los 80.

PDA's para reconocimiento de lo que se escribe en pantalla, sistemas de reconocimiento de voz e incluso sistemas de reconocimiento de imágenes (óptico).

Se debe tener en cuenta que, aunque las redes neuronales son muy útiles en caso de problemas de lógica difusa, no dan solución a todos los problemas, ya que no son nada útiles, por ejemplo, en casos de problemas numéricos clásicos con los que estamos tan acostumbrados a encontrarnos, como es por ejemplo solucionar un ecuación.

Los ordenadores de 5ª generación podrían basarse en procesadores ópticos

GENERALIDADES DE LA I.A.

Tras todo lo explicado se puede decir que un sistema inteligente debe cumplir las siguientes normas de tipo general:

1- Debe poder captar generalidades: Se le tienen que poder detallar situaciones complejas de forma global no dando datos individualmente, sino dando información interrelacionada que forme un todo dentro de la situación.

2- Facilidad de comunicación: El sistema debe estar diseñado de forma que no sea difícil darle los datos que se deseen sobre cualquier tema, siendo el ideal un sistema de reconocimiento de voz oral unido a un sistema de interpretación de lenguaje natural.

3- Debe ser capaz de reducir los problemas: El sistema, una vez tenga toda la información que se le haya proporcionado del problema, debe ser capaz de simplificarlo antes de solucionarlo. Por ejemplo, si se le pidiera que buscase el camino más corto entre dos ciudades del globo, el programa debería ser capaz de buscar un camino corto de forma inteligente, y no dar con la solución a base de comparar todas las rutas posibles que hayan por las carreteras de todo el planeta y dar la más corta que haya encontrado.

4- Adaptable a situaciones: El sistema debe poder adaptarse a muchos tipos de problemas, aunque para ello pierda fiabilidad y precisión en los resultados.

5- Modificable: Que el sistema pueda autoreprogramarse fácilmente en caso de que se le indiquen modificaciones que debe adoptar en su metodología.

RESUMEN

Con todo lo explicado, el programador tiene una referencia global de todos los aspectos de la I.A, tanto en datos históricos como en lo que a técnicas de programación se refiere. A partir de aquí, el lector puede investigar en el área que más le interese de la Inteligencia Artificial, puesto que ésta posee tantos campos y puntos de partida como se pueda imaginar.

BIBLIOGRAFÍA

·Real brains. Artificial Mids. J.L. Casti y A.Karlqvist (eds.). 1987 Norton Holland

·An introduction to computing with Neural Nets. R.P. Lippman. IEEE ASSP Magazine. 4.4. 1987

·Nuevas tecnologías. Biblioteca de Electrónica/Informática. Orbis. Marcombo. Nº60.

·The image processing Handbook. John C Russ. CRC Press. 1992

·Vision in Man and Machine. Martin D.Levin. McGraw Hill. 1995

·Computers & Thought. Feingenbaum y Felman. Mc Graw Hill. 1963

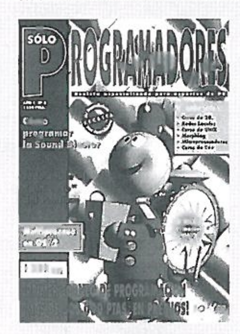
·Machines who think. P Mc Corduck. Freeman. 1979.

·Optical Logics in the light of computer technology. R.W.Keyes. optical Acta. 32, 525. 1985.

CÓMO SUSCRIBIRSE A

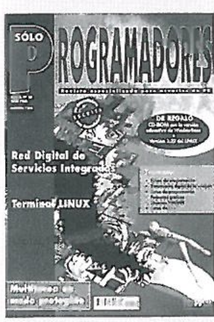








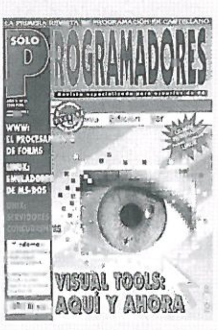












Suscríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos...Domicilio..

Población..C.P ....Provincia.........................Telf...............................Profesión.....................

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº _ _ _ _   _ _ _ _   _ _ _ _   _ _ _ _

Fecha de caducidad de la tarjeta................Nombre del titular, si es distinto..

☐ Domiciliación bancaria.

Señor Director del banco ....

Población ....

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número...

el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L. como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

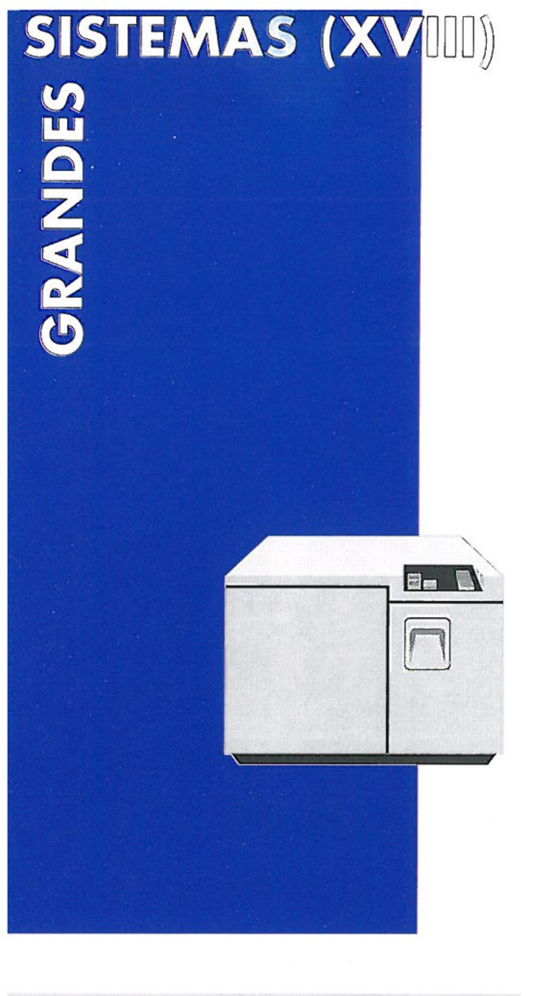
Firma:

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)



UTILIDADES MVS (III)

José María Peco

El presente artículo cierra esta miniserie dedicada a las utilidades del MVS. Son muchas e importantes las utilidades que se quedan en el tintero, pero el objetivo que se había marcado con este tema sólo era comentar las 'utilities' de uso más frecuente, así como resaltar la idea de que un desarrollador debe disponer de forma fácil y accesible de todos los manuales de los productos instalados, sin necesidad de recurrir a la fotocopia de apuntes manuscritos de otro compañero.

Como todo informático conoce, el MVS es el sistema operativo para grandes procesadores de IBM, estando justificada la existencia de estos grandes procesadores por la necesidad de gestionar grandes volúmenes de datos.

En los dos artículos anteriores se han comentado algunas de las muchas utilidades estándar con que cuenta una instalación de este tipo, pues este S.O. se encuentra acompañado por un conjunto de programas que, sin formar parte del mismo, facilitan las labores básicas de gestión de ficheros, tales como borrar, imprimir, copiar, etc..., lo cual, por otra parte, cabe comentar que es algo normal, y que ocurre con cualquier sistema operativo.

Las utilidades vistas hasta ahora provienen incluso de la versión anterior del MVS, concretamente del OS/VS, pero que con objeto de mantener la migración de aplicaciones a las versiones nuevas han evolucionado junto con él.

Los programas que se van a tratar en este artículo no son simples programas de utilidad como los vistos anteriormente, sino que son unas 'facilities' o utilidades integradas y más completas (aunque no más complejas) pero que también son de gran utilidad para el desarrollador:

- **IDCAMS:** Es una utilidad definida inicialmente para manejar objetos VSAM y que, formando parte del DFP (Data Facility Product), permite utilizar todos los métodos de acceso, excepto los ficheros particionados.

- **IKJEFT01:** Es el propio TSO (Time Sharing Operation).

- **PRINTDS:** Comando de TSO para imprimir ficheros.

- **ISRSUPC:** Programa que permite buscar cadenas de caracteres entre los distintos miembros de una librería o fichero PDS.

VERSIONES DEL MVS

Con el fin de enmarcar en el tiempo la evolución del MVS (Multiple Virtual Storage), se incluye este apartado que enumera las distintas versiones del mismo, resaltando el hecho de que son las diferentes versiones de los productos que acompañan al MVS básico las que establecen las diferentes versiones de este S.O.

Así, cuando se habla de:

- La versión 1 de MVS (1972), con sus diferentes revisiones (releases), se esta hablando de lo que comúnmente se conoce como MVS/370, pero se corresponde con el OS/VS1 versión 2, e introduce el concepto de Espacio de direcciones.

- La versión 2 de MVS (1980), más sus diferentes releases componen el MVS/XA. (Extended Architecture).

Ésta realmente fue una revisión de la versión anterior para solucionar la gran cantidad de problemas presentados en aquella.

El MVS/XA es una versión que afecta tanto al hardware como al software, ya que la memoria virtual pasa de 16 Mb a 2 Gb, necesitando 4 bytes para direccionar cada posición de memoria. (ver en núm.18 de Sólo Programadores Campos de la PSW : pag. 43)

- La versión 3 de MVS (1988), se corresponde con lo que se conoce como

MVS/ESA (Enterprise System Architecture).

Esta versión se diferencia de la anterior, entre otras cosas, en que puede direccionar hasta un máximo de 15 espacios de direcciones de 2 Gb. Así mismo esta versión introduce el uso de microcódigo.

En futuros artículos se comentarán los distintos conceptos que se mencionan en esta breve reseña histórica del MVS.

IDCAMS

La 'Facility' IDCAMS (Access Method Services) es una utilidad proporcionada por IBM para controlar objetos VSAM, tales como catálogos, paths, clusters, índices alternativos, etc. (ver Sólo Programadores: Ficheros VSAM), aunque también puede ser usado para manejar todos los tipos de ficheros a excepción de los PDS o ficheros particionados.

Con objeto de mantener la migración de aplicaciones a las versiones nuevas, las utilidades evolucionaron junto con el MVS

Desde el punto de vista del JCL no se diferencia en nada de las utilidades vistas, pues utiliza las mismas Ddnames SYSPRINT y SYSIN.

Los comandos admitidos por esta utilidad a través de la SYSIN se podrían clasificar en:

- Creación de catálogos.

Los ficheros VSAM no pueden ser creados desde un JCL, sino que se hacen mediante comandos AMS, y como se sale del objetivo de este articulo, no se trata este punto.

- Definición de ficheros GDG (Generation Data Group). Estos son ficheros secuenciales normales que sólo tienen de especial el nombre ya que bajo un mismo nombre se pueden referenciar varios ficheros.

La definición de un fichero GDG consta de dos pasos: En el primero se crea lo que es el nombre del fichero en el catálogo. La figura 1 muestra un paso de JCL para realizar este cometido. Como puede verse en él, no se especifican características, como ocurre cuando se crea cualquier fichero, ya que sólo es una entrada en el catalogo de usuario, no en el VTOC pues todavía no existe físicamente.

Una vez creado el nombre, cuando se desee añadir una versión a ese fichero, se realizará lo siguiente:

- Añadir el sufijo (+1) al nombre.

- Especificar las características de este fichero secuencial, que no tienen por qué coincidir con las otras versiones.

- Poner como disposición inicial NEW.

En cambio, cuando se desee leer una versión habrá que especificar como nombre el nombre del GDG y el sufijo (0) para la ultima versión, (-1) para la penúltima, etc..., y como disposición inicial hay que especificar SHR (Shared compartido.Ver Sólo Programadores núm. 20)

Figura 1
Esqueleto para crear un GDG

```
//XXXXXXDG JOB (12,345),'prueba',NOTIFY=XXXXXX,CLASS=x,MSGCLASS=u
//** ************************************************************ **
//PASO1    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DEFINE GDG (NAME(JMPDES.EJEMPLO.GDG) LIMIT(5) NOEMPTY SCR)
/*
```

El formato del comando que define un GDG es el siguiente:

DEFINE GDG (-
NAME(nombre_gdg)-
LIMIT(num_ver) -
EMPTY/NOEMPTY -
SCRATCH/NOSCRATCH)

donde:
NAME sirve para especificar el nombre del fichero GDG a definir

nombre_gdg representa el DSN del fichero.

LIMIT sirve para especificar el número de versiones que se desea mantener.

num_ver indica el número de versiones de 1 a 155.

EMPTY (opcional) especifica que se descataloguen todos los nombres de fichero cuya versión supere la especificada en LIMIT.

NOEMPTY sirve para especificar que según se vayan generando versiones nuevas, se vayan borrando las mas antiguas, de forma que sólo estén activas las últimas *LIMIT*, tal y como muestra la figura 2, pudiendo ver el listado de la entrada del catalogo (no tiene VOLUMEN) y las 5 versiones (6 a 10) qu se especificaron en la figura 1.

Figura 2
Nombres de las versiones 6 a 10 de un GDG

```
DSLIST - DATA SETS BEGINNING WITH JMPDES.E*.G* ------------------ ROW 1 OF 7
COMMAND ===>                                                   SCROLL ===> CSR

COMMAND      NAME                                       MESSAGE          VOLUME
-------------------------------------------------------------------------------
             JMPDES.EJEMPLO.GDG                                          ??????
             JMPDES.EJEMPLO.GDG.G0006V00                                 X1DES91
             JMPDES.EJEMPLO.GDG.G0007V00                                 X1DES91
             JMPDES.EJEMPLO.GDG.G0008V00                                 X1DES91
             JMPDES.EJEMPLO.GDG.G0009V00                                 X1DES91
             JMPDES.EJEMPLO.GDG.G0010V00                                 X1DES91
***************************** END OF DATA SET LIST ****************************
```

SCRATCH (opcional) especifica que se borre fisicamente el fichero cuando se descatalogue.

- Listar entradas del catalogo.

Figura 3
Ejemplo de jcl para ejecutar el comando de TSO: PRINTDS

```
****** ***************************** TOP OF DATA ******************************
000001 //XXXXXX6V JOB  (30,000),'SOTANO-1 SALA-2',CLASS=T,MSGCLASS=X,
000002 //             NOTIFY=XXXXXX,MSGLEVEL=(0,0)
000003 //V158         OUTPUT CHARS=GTE5,PAGEDEF=388STD,FORMDEF=STDA
000004 //*=================================================================
000005 //*                     IMPRIMIR TEMPORAL CON MARGEN
000006 //*=================================================================
000007 //*
000008 //PASOASEC  EXEC PGM=IKJEFT01
000009 //*
000010 //SYSTSPRT  DD SYSOUT=*
000011 //SYSTSIN   DD *
000012    PR DATASET('JMPDES.SALIDA')     +
000013       COLUMNS(1:80) PAGELEN(80) LMARGIN(10) ALL +
000014       DEST(LOCAL) OUTDES(V158)
000015 /*
```

El comando LISCAT permite listar entradas del catálogo. Su formato es el siguiente:

LISCAT CATALOG (nom_catalogo).

• Copiar ficheros.
El comando *REPRO* permite realizar las siguientes copias de ficheros:

- Copiar ficheros secuenciales a ficheros VSAM.

- Copiar ficheros VSAM a ficheros VSAM.

- Copiar ficheros VSAM a ficheros secuenciales no-VSAM.

- Copiar ficheros ISAM (Indexed Sequential Access Method) a ficheros VSAM.

El esqueleto de un paso que realice esta función, sería:

//COPIAR EXEC PGM=IDCAMS
*//SYSPRINT DD SYSOUT=**
//ENTRADA DD DSN=nombre_fichero_entrada...
//SALIDA DD DSN=nombre_fichero_salida ...
*//SYSIN DD **
REPRO -
INFILE(ENTRADA) OUTFILE(SALIDA) COUNT(nnn)
*/**

Donde ENTRADA y SALIDA son las Ddnames que tienen asociados los nombres de los ficheros de entrada y de salida y nnn representa el numero de registros que se desean copiar. En el caso de no especificar el parámetro COUNT se entiende que es todo el fichero.

• Imprimir ficheros.
El comando PRINT permite imprimir tanto ficheros VSAM como ficheros NO-VSAM

El esqueleto de un paso que realice esta función es el siguiente:

//COPIAR EXEC PGM=IDCAMS
*//SYSPRINT DD SYSOUT=**
//ENTRADA DD DSN=nombre_fichero_entrada...
*//SYSIN DD **
PRINT -
INFILE(ENTRADA) formato
*/**

Donde *ENTRADA* representa la Ddname que tiene asociado el nombre del fichero a imprimir y formato puede tomar uno de los siguientes valores:

- *CHAR:* imprime los registros en formato carácter.

- *HEX:* Imprime los registros en formato hexadecimal.

- *DUMP :* Imprime los registros en ambos formatos.

• Borrar ficheros.
comando *DELETE* permite borrar tanto ficheros VSAM como ficheros NO-VSAM.

El esqueleto de un paso que realice este comando es el siguiente:

//COPIAR EXEC PGM=IDCAMS
*//SYSPRINT DD SYSOUT=**
//FICHERO DD DSN=nombre_fichero...
*//SYSIN DD **
DELETE nom_fichero -
FILE(FICHERO) -
PURGE -
ERASE -
SCRATCH -
CATALOG(nom_catalogo)
*/**

Este comando admite que el fichero a borrar se especifique directamente, bien a través del parámetro FILE especificando en el nombre de la Ddname que tiene asociado.

El resto de parámetros es opcional, significando lo siguiente:

La versión 2 de MVS realmente fue una revisión de la versión anterior para solucionar la gran cantidad de problemas presentados en aquella

PURGE: Borra la entrada mirando la fecha en que expira el fichero.

ERASE: Borrar el fichero y sobreescribe ceros binarios.

SCRATCH: Se omite para ficheros VSAM.

CATALOG: Nombre del catalogo VSAM del que se desea borrar el ítem.

• *SET MAXCC=nn*
Este comando espoecifica que, independientemente de cuál haya sido el resultado de la ejecución anterior, devuelva como código de retorno el valor nn especificado.

Si todos los parámetros de un comando no caben en una línea, éstas se pueden unir especificando el carácter '+' al final de la misma

IKJEFT01

El tema que se está tratando es lo suficientemente amplio como para incluir en él a IKJEFT01, aunque al igual que en el punto anterior, el programa que pone título a este apartado no sea realmente una utilidad, sino toda una facility o conjunto de programas que favorecen el desarrollo, pues es el propio TSO.

Por otra parte, como el autor entiende que este programa se merece algo más que un apartado, sólo va a comentar las Ddnames necesarias para ejecutar este programa en batch, lo cual permitirá:

- ejecutar comandos de TSO.
- submitir JCLs.
- Enviar mensajes a otros usuarios.
- etc.

• *SYSTSPRT.* Esta Ddname se corresponde con la SYSPRINT de las utilidades vistas anteriormente, y define el destino al que se dirigirán los mensajes generados por el programa.

• SYSTSIN. Se corresponde con la SYSIN y contiene los comandos que serán ejecutados por el programa. En el ejemplo de la figura 3, el comando PRINTDS.

PRINTDS o PR

Realmente, no se puede decir que se ajuste exactamente a la definición de utilidad dada en el primer artículo, ya que es un comando de TSO, pero se incluye por no ser muy conocido en las instalaciones a nivel de desarrollador y, además, ser uno de los mas útiles de TSO pues, como su nombre indica, hace un PRINT de un data set (DS) o fichero, ya sea éste secuencial o particionado.

Entre sus características, cabe destacar las siguientes:

- Escribe cabeceras en cada pagina.
- Se puede establecer margen superior (TMARGIN), izquierdo (LMARGIN), espaciado.
- Clase (CLASS) e impresora de salida (DEST).
- Se puede direccionar la salida a un fichero (TODATASET).
- En el caso de ficheros particionados, genera un índice al final del listado.
- Se puede listar solo el directorio de un fichero particionado.

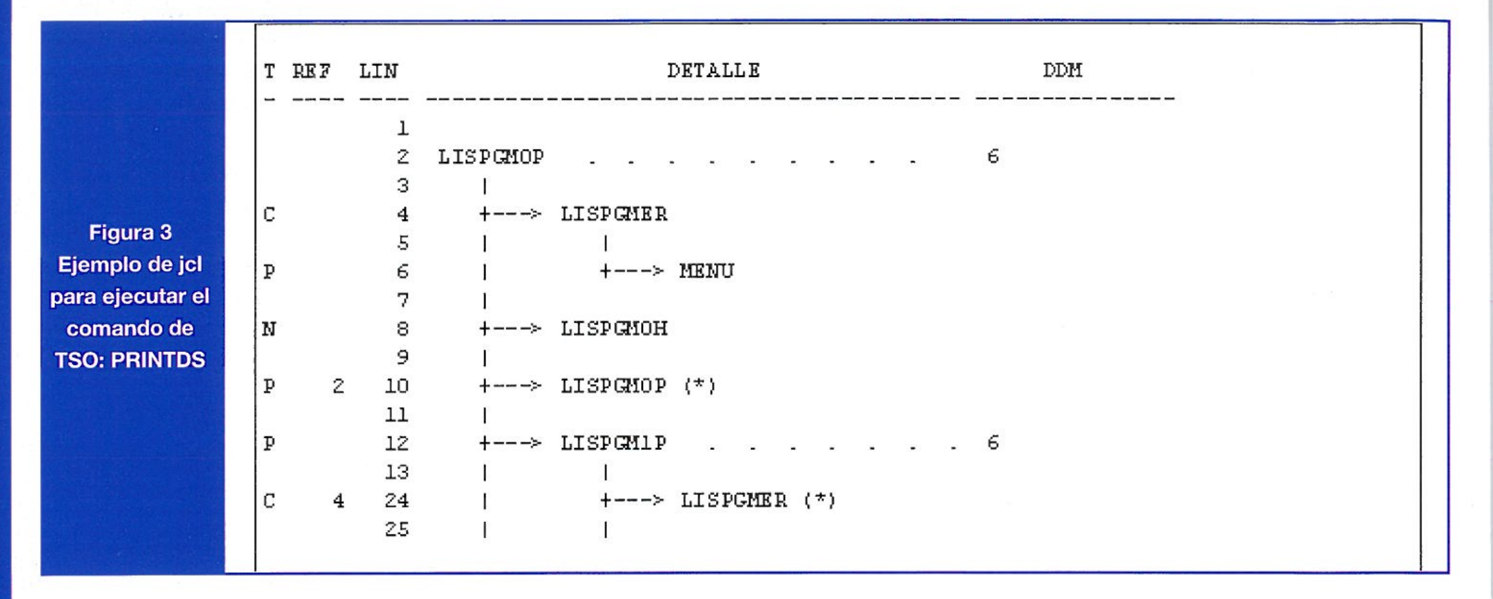
Para conocer la sintaxis y todas las opciones aceptadas por este potente comando, basta ejecutar TSO HELP PRINTDS.

Para ejecutar on-line desde TSO sólo hay que especificar desde la línea de comando TSO PRINTDS DSNAME('JMPDES.JMP.SALIDA') DEST(RMT101).

Para que el lector pueda tener un ejemplo completo del uso de este comando, la figura 3 muestra el paso de un JCL que ejecuta el programa IKJEFT01, es decir TSO, y en la entrada correspondiente a los comandos a ejecutar incluye el comando PRINTDS y la serie de parámetros necesarios para su ejecución.

Figura 4
Ejemplo de jcl para buscar la cadena PRINTDS en la librería SYS.HELP

```
00010 //XXXXXXST JOB  (12,345),'scan tso',CLASS=T,MSGCLASS=X,
00020 //              NOTIFY=XXXXXX,MSGLEVEL=(0,0)
00030 //*****************************************************************
00040 //*             BUSCAR CADENA EN DIRECTORIO
00050 //*****************************************************************
00060 //BUSCAR EXEC PGM=ISRSUPC,PARM=(SRCHCMP,'ANYC')
00070 //NEWDD  DD DISP=SHR,DSN=sys1.help
00080 //OUTDD  DD SYSOUT=*
00090 //SYSIN  DD *
00100   SRCHFOR 'printds'
00110 /*
```



```
T REF  LIN                    DETALLE                    DDM
- ---- ---- --------------------------------------- --------------
        1
        2  LISPGMOP  . . . . . . . . . .              6
        3     |
C       4     +---> LISPGMER
        5     |        |
P       6     |        +---> MENU
        7     |
N       8     +---> LISPGMOH
        9     |
P    2 10     +---> LISPGMOP (*)
       11     |
P      12     +---> LISPGM1P  . . . . . . . .         6
       13     |        |
C    4 24     |        +---> LISPGMER (*)
       25     |        |
```

Figura 3
Ejemplo de jcl para ejecutar el comando de TSO: PRINTDS

Es de resaltar que cada línea especifica un comando, pero si todos los parámetros de un comando no caben en una línea, éstas se pueden unir especificando el carácter '+' al final de la misma.

SYS1.HELP

Para cerrar el tema, se incluye la mención a esta librería propia del sistema, (tiene por primer cualificador SYS1), ya que en ella se puede encontrar la sintaxis de muchos comandos y programas del sistema, así como la explicación de cada uno de los parámetros admitidos por el comando y documentación de los mensajes de error que generan.

Como ejemplo de lo dicho en el párrafo anterior, se aconseja listar el miembro IKJEFY55, que muestra todo el HELP asociado al comando PRINTDS, visto en el apartado anterior.

Pero, como a veces se desconoce el nombre del comando a ejecutar, no se puede pedir al sistema que muestre la ayuda asociada. Por esta razón se recomienda el uso del programa ISRSUPC, propio del sistema, con los parámetros y Ddnames que se muestran en la figura 4, el cual permite buscar una palabra, por ejemplo "MARGIN" entre los miembros de una librería, obteniendo de esta forma la relación de comandos que usan dicha palabra bien en su sintaxis, bien en el texto de ayuda que acompaña a dicho comando.

Como a veces se desconoce el nombre del comando a ejecutar, no se puede pedir al sistema que muestre la ayuda asociada

BIBLIOGRAFÍA

MVS manual para programadores
ISBN : 84-481-0092-1
Editorial : McGraw-Hill

EXPERT MVS/ESA JCL a guide to advanced Techniques
ISBN: 0-07-009820-4
Editorial : McGraw-Hill

SYSTEM/370 Job Control Language
Autor: Gary DeWARD BROWN
Editorial: JOHN WiLEY&SONS

TSO para desarrolladores (JCL/CLIST/ISPF)
Num.RPI: M-33933
Inédito
Autor: José María Peco (En el disco que acompaña a la revista se encuentra el fichero LIBRO.DOC, que contiene el índice y una presentación del mismo junto con un browser freeware de Microsoft para documentos WORD).

UTILIDAD

La utilidad que acompaña este mes al artículo retoma el JCL que acompañaba a la primera entrega de este tema, para mostrar el resultado que se obtiene al ejecutar una utilidad desarrollada por el autor de este artículo y que tiene por finalidad listar los objetos natural cuyo nombre figura en un fichero que sirve de entrada y que se representa en dicho esqueleto con la variable &ENTRADA.

Puesto que el tema que nos ocupa no es NATURAL, no se comentan los programas invocados para ese entorno. Sólo se acompaña dicho JCL para que se pueda tener un ejemplo real del uso de las distintas utilidades tratadas en este tema.

Uno de los resultados obtenidos con la ejecución de este JCL es el árbol jerárquico de objetos que se muestra en la figura 5, correspondiente al Browser de NATURAL, que se distribuyó con los artículos del tema anterior: Natural.

Por último, desde estas líneas se lanza un mensaje a nuestros lectores para que, aquellos que quieran ver publicadas las utilidades desarrolladas por ellos mismos, las remitan junto con una breve descripción de la misma a la redacción de la revista o bien a la dirección del autor: <jmpeco@luznet.es>. También pueden remitir aquellas direcciones y foros de INTERNET que consideren interesantes y relacionados con el tema de Grandes Sistemas.



OPTIMA ++ 1.5 PROFESSIONAL

Fernando de la Villa

Muchos desarrolladores de aplicaciones C++ bajo Windows llevan tiempo quejándose de la complejidad de las herramientas de desarrollo disponibles en el mercado. Por otro lado, las herramientas RAD existentes usan lenguajes de programación que en cierto tipo de aplicaciones no proporcionan la potencia suficiente. La solución es una postura intermedia que pueda satisfacer las necesidades de desarrollo rápido de aplicaciones, a la vez que proporcione un lenguaje potente. Optima++ es la respuesta.

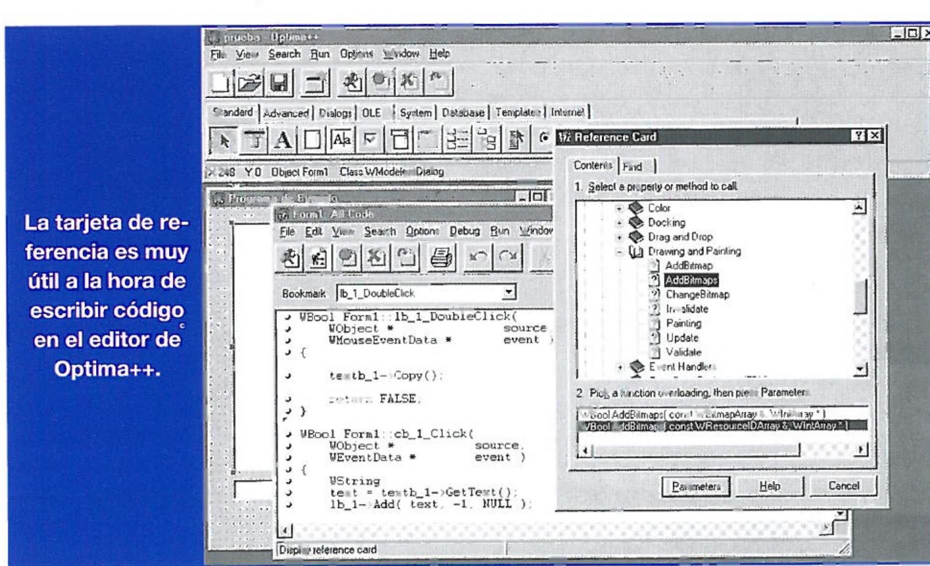
En pocas líneas, Optima++ es una herramienta de desarrollo rápido de aplicaciones bajo Windows con el lenguaje C++ que recoge conceptos de la denominada programación visual, así como de la programación orientada a eventos.

NOVEDADES

La nueva versión incluye diversas novedades con respecto a Optima++ 1.1, además de las omnipresentes correcciones y pequeños cambios que siempre se realizan entre las distintas versiones de un producto. Entre las novedades destacan la incorporación de nuevos controles, soporte para aplicaciones orientadas a Internet, manejadores nativos de bases de datos y soporte para herramientas de revisión y control.

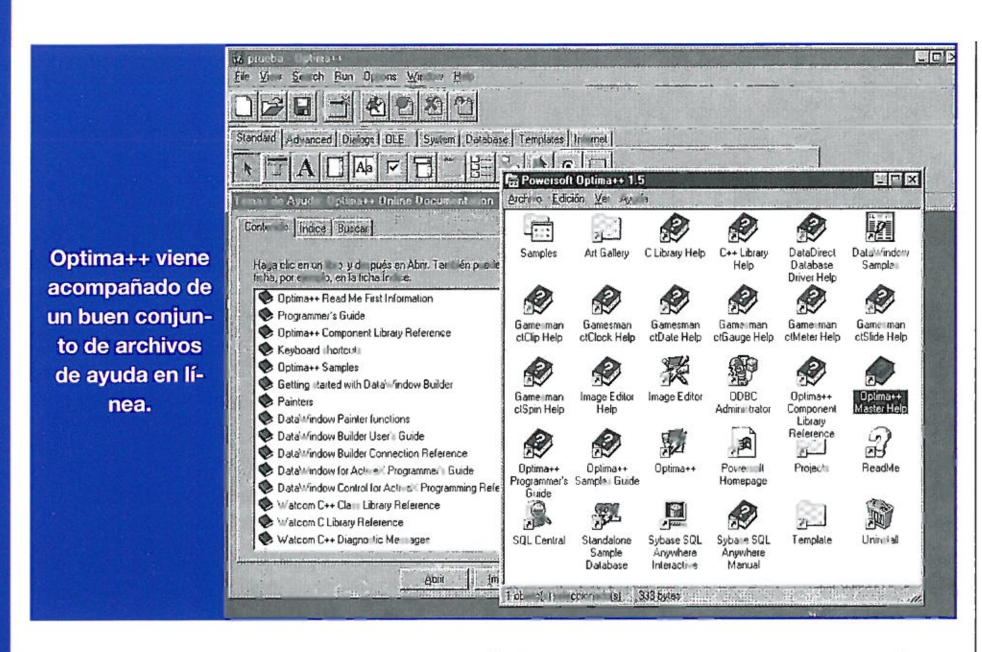
Uno de los nuevos controles incluidos es el denominado *DataWindow*. Este control permite visualizar información procedente de una base de datos de forma sencilla y atractiva. Otro de los controles es la rejilla (*grid*) y se utiliza para mostrar datos en forma tabular, de forma similar a como lo hacen las hojas de cálculo y bases de datos tradicionales. Por último, la caja de texto enmascarada proporciona entrada de texto mejorada a través del teclado. Es ideal para datos con un formato predefinido, como el DNI, las fechas y los teléfonos.

En cuanto a las mejoras relacionadas con Internet, Optima++ permite que las aplicaciones se comuniquen con un servidor web mediante los interfaces CGI, ISAPI o NSAPI. Además, incluye clases que facilitan el acceso a los servicios proporcionados por WINSOCK y WINSOCK32, facilitando el desarrollo de aplicaciones para Internet.



La tarjeta de referencia es muy útil a la hora de escribir código en el editor de Optima++.

La calidad de las aplicaciones orientadas al RAD (Desarrollo Rápido de Aplicaciones) está mejorando notablemente. Optima++ es, sin duda, uno de los máximos exponentes de esta filosofía de creación de aplicaciones.



Optima++ viene acompañado de un buen conjunto de archivos de ayuda en línea.

Para un mejor seguimiento del código fuente de cada proyecto se ofrece un interfaz con diversos sistemas de control y revisión, como MKS RCS, INTERSOLV PVCS y ObjectCycle. También permite la utilización de cualquier sistema compatible con el interfaz SCC de Microsoft.

La mayoría de las características comentadas sólo están disponibles en las ediciones Professional y/o Enterprise del producto.

EDICIONES

Existen tres ediciones diferentes de Optima++ : Developer, Professional y Enterprise. La primera es la más sencilla y está pensada para los desarrolladores que no necesiten las características avanzadas de las demás ediciones. Enterprise es la edición más profesional y contiene todos los elementos que componen Optima++. La edición Professional, comentada en estas páginas, en la solución intermedia. Algunas de las características más interesantes comunes a las tres ediciones son:

- Compilador 32 bits de código nativo con optimizaciones basado en la prestigiosa tecnología de compilación Watcom, utilizada también en Watcom C++.
- *Visual SQL Query Builder*: se emplea para la creación automática de sentencias SQL.
- Soporte para aplicaciones multi-hilo.
- Conexión directa con manejadores ODBC.
- Integración de componentes ActiveX.

Además de las características mencionadas en el epígrafe anterior, la edición profesional incluye:

- Cerca de 220 componentes y clases.
- *DataPipeline*: se trata de una utilidad de migración de datos.
- Interfaz de programación en grupos.

También se ha incluido una versión limitada de Sybase SQL Anywhere en el paquete de Optima++ Professional. Se trata de un potente gestor de bases de datos basado en SQL, el lenguaje estándar de facto para la gestión y el acceso a bases de datos. Dispone principalmente de un motor de base de datos, un servidor y un cliente. Con todo ello es posible construir un sistema de base de datos en red, donde el servidor pone los datos a disposición de las aplicaciones creadas con Optima++. En el CD de Sólo Programadores del pasado mes se encuentra una versión trial de este gestor de bases de datos.

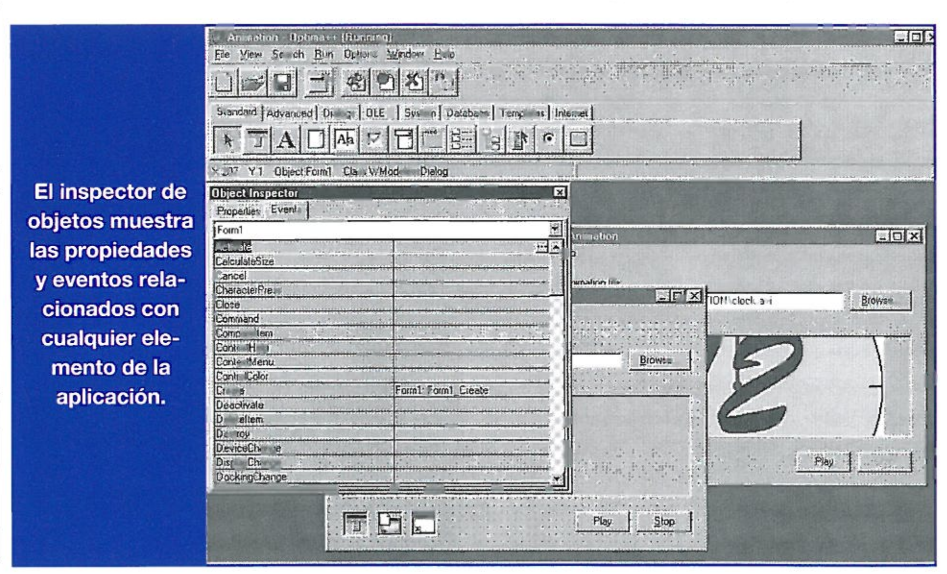
REQUERIMIENTOS

Para ejecutar con éxito la herramienta no se necesita una gran plataforma hardware. Basta con una máquina PC o compatible equipada con un procesador 486 o superior y 16 megas de RAM como mínimo. El ordenador debe tener instalado previamente el sistema operativo Windows NT 3.51 (o superior), o bien Windows 95. Se necesitan, al menos, 40 megas libres en el disco duro, aunque el espacio total depende en gran medida de la edición (Developer, Professional o Enterprise) y de los componentes elegidos. Una instalación normal ocupará unos 100 megas, mientras que una completa puede incrementar esta cifra hasta 350 megas. Además, se recomienda un fichero de intercambio (swap) de 40 megas para trabajar con soltura.

La distribución de Optima++ se realiza en CD-ROM, por lo que es imprescindible una unidad lectora de compactos.

OPTIMA++ EN LA PRÁCTICA

La mecánica de creación de aplicaciones con Optima++ es muy similar a la



El inspector de objetos muestra las propiedades y eventos relacionados con cualquier elemento de la aplicación.

de otras populares herramientas de desarrollo, como Visual Basic y Delphi. Esta forma de trabajo se está extendiendo entre los programadores a un ritmo vertiginoso, debido en gran parte a que facilita enormemente el diseño e implementación del interfaz de usuario. Ha demostrado ser una manera sencilla y rápida de minimizar el tiempo empleado en la creación de aplicaciones para Windows.

Para aquellos que todavía no lo conozcan, el método consiste en crear la aplicación alrededor de los denominados formularios, que no son más que ventanas inicialmente vacías. En los formularios se van colocando los controles necesarios. Existen multitud de controles útiles para una aplicación, como por ejemplo, los botones, las listas, las cajas de diálogo y un largo etcétera. Los controles y formularios tienen unas propiedades asociadas que definen sus características y comportamiento en el programa. Además, pueden responder a una serie de eventos como puede ser una pulsación con el ratón. Estos eventos tienen código asociado y desencadenan acciones en el programa.

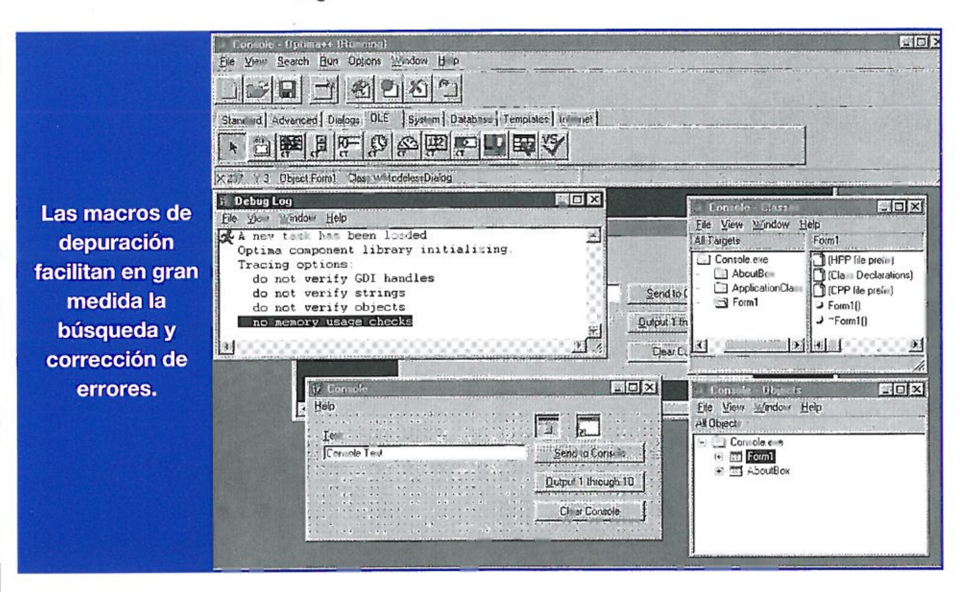
Optima++ dispone de una paleta de controles ordenados por tipo. Para situar un control en el formulario correspondiente basta con seleccionar el control en la paleta y a continuación colocarlo en el lugar adecuado. A partir de ese momento se puede acceder a las funciones relacionadas con el mismo a través de un menú de contexto que aparece pulsando con el botón derecho del ratón sobre el control. De este modo se pueden editar las propiedades y el código de los eventos, además de otras útiles funciones como el ajuste del tamaño y la alineación con otros controles.

La introducción de código en Optima++ se puede realizar de una forma realmente visual. Consiste en arrastrar un control situado en el formulario al punto de la ventana de edición donde se quiera añadir código relacionado con dicho objeto. Nada más soltar el objeto aparece la tarjeta de referencia. Se trata de una ventana con todas las propiedades y métodos posibles para el objeto. Pinchando con el ratón sobre cualquiera de ellos, se tiene la opción de modificar los correspondientes parámetros, y se añade el código automáticamente al programa. Este sistema facilita enormemente a los principiantes la escritura de código y ayuda a conocer rápidamente las posibilidades del producto.

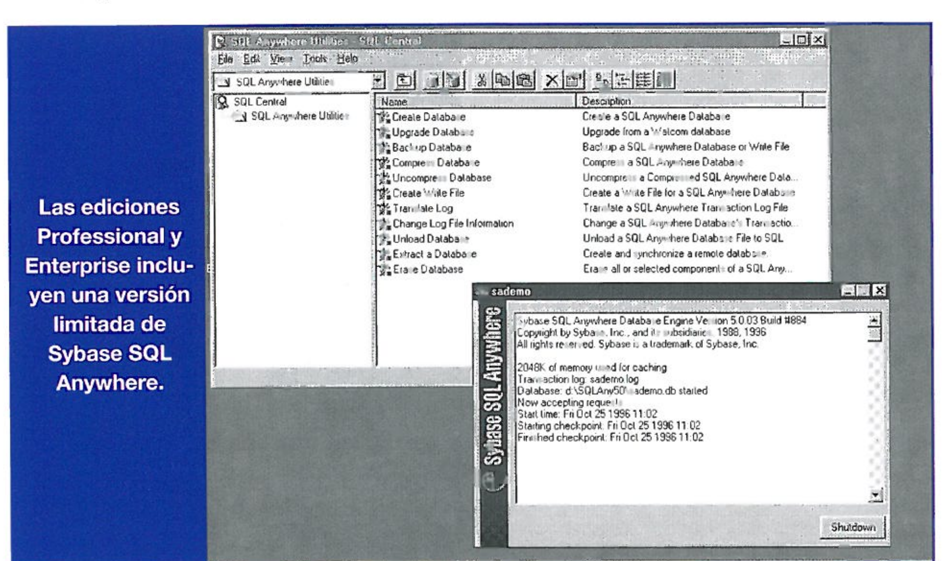
Optima++ incorpora un entorno de desarrollo sencillo y potente a la vez

Los más experimentados seguramente prefieran la forma clásica de escribir un programa. Por supuesto, también es posible esta opción, aunque hay dos posibilidades. La ventana de edición puede mostrar todo el código asociado con un formulario, o bien solamente el código correspondiente a un evento. Este último método es el utilizado habitualmente en otros productos. Optima++ ofrece un editor de código repleto de opciones. Permite la utilización de marcadores para retornar a las partes más interesantes del código, soporta código de sólo lectura y admite el uso de colores con el fin de aumentar la legibilidad.

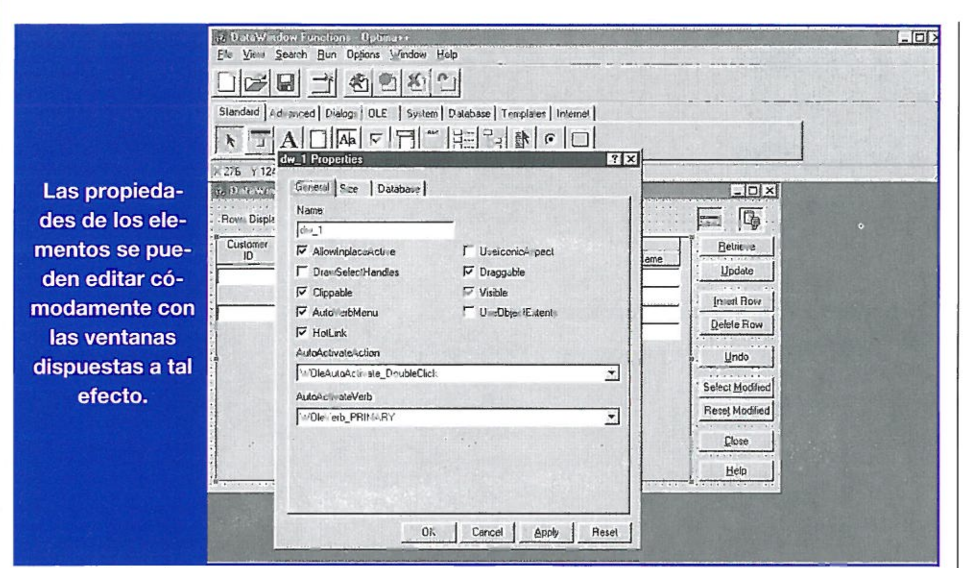
La depuración de programas también resulta bastante sencilla y completa gracias a las numerosas opciones disponibles. La forma habitual de depurar una aplicación es mediante puntos de ruptura (*breakpoints*), que permiten examinar el estado del programa en un momento concreto de la ejecución del mismo, pero existen otras formas. Optima++ incluye un interesante mecanismo de depuración mediante macros



Las macros de depuración facilitan en gran medida la búsqueda y corrección de errores.

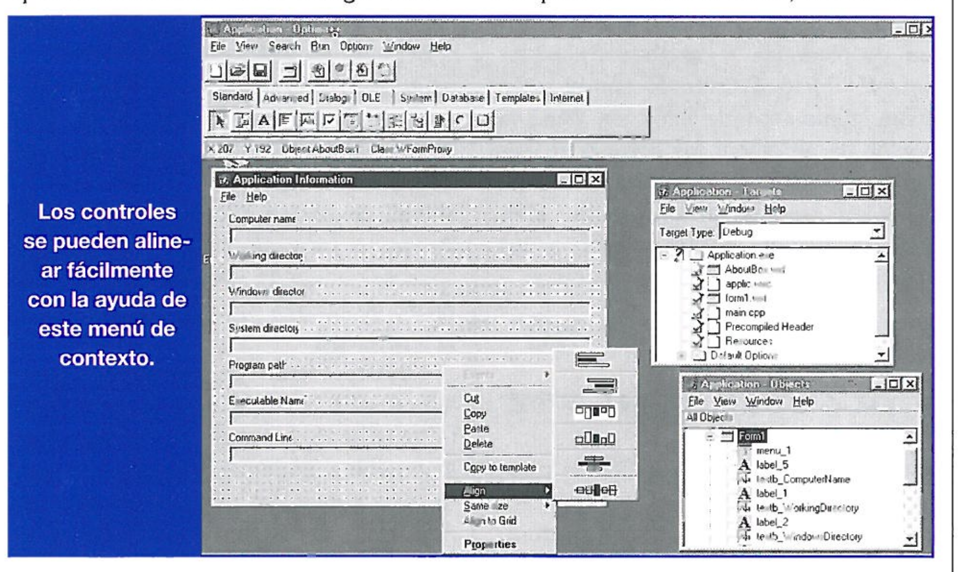


Las ediciones Professional y Enterprise incluyen una versión limitada de Sybase SQL Anywhere.



Las propiedades de los elementos se pueden editar cómodamente con las ventanas dispuestas a tal efecto.

que se introducen en el código fuente del programa. Se pueden utilizar aserciones, avisos, validación de objetos y visualización en una ventana especial de depuración. Casi todas estas macros sólo están activas durante el desarrollo de la aplicación. Cuando se genera la versión definitiva del programa desaparecen automáticamente.

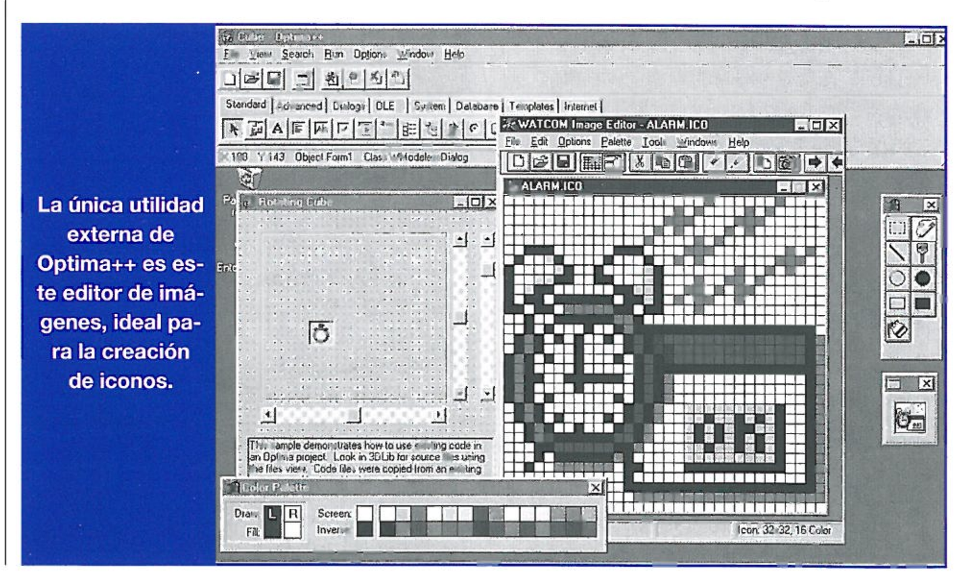


Los controles se pueden alinear fácilmente con la ayuda de este menú de contexto.

Las aserciones evalúan una condición y, si el resultado es cierto (cualquier valor distinto de cero), el programa continua su ejecución. En caso contrario, el programa para y se puede examinar el estado del mismo. Existe otra versión de esta aserción capaz de mostrar, además, una cadena de texto informativo. Las dos macros comentadas anteriormente disponen de una versión que sí permanece en el programa definitivo. Se utilizan exclusivamente para evitar males mayores en caso de producirse una situación excepcional. Por otro lado, las macros de aviso simplemente muestran un cuadro de diálogo con un aviso útil para el programador. Otra macro interesante es la de validación de objetos en tiempo de ejecución. De este modo, se puede asegurar que un objeto ha sido construido correctamente y que no ha sido destruido de forma accidental. Por último, es posible mostrar mensajes, valores de variables y resultados de operaciones en la ventana de depuración mientras se está ejecutando el programa. En este caso, no es necesario parar la ejecución del programa para verificar el correcto funcionamiento del mismo.

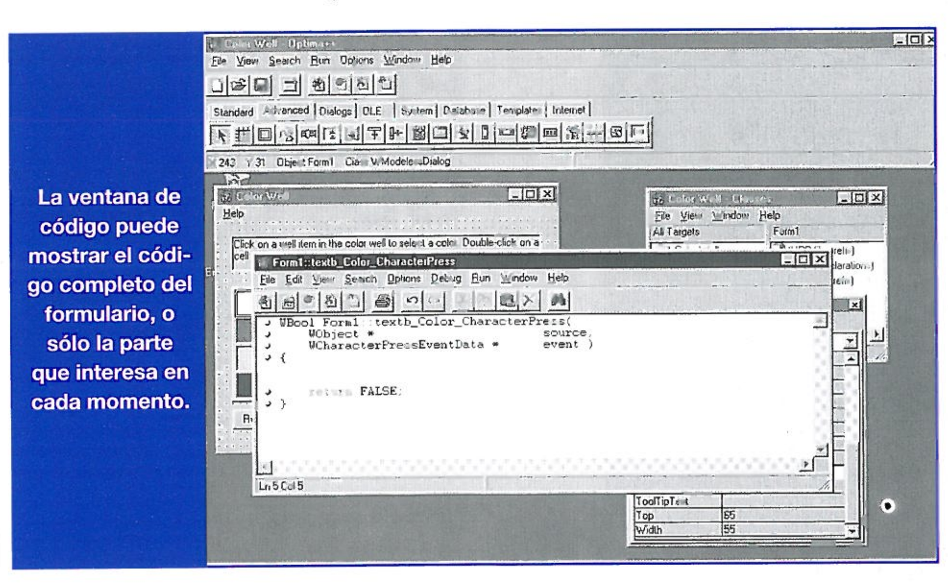
Existen numerosas ventanas orientadas a la depuración. Se puede ver la pila de llamadas realizadas entre los distintos módulos del programa, el código en ensamblador, la pila del sistema y los registros de la CPU y del coprocesador. También se dispone de ventanas para realizar un seguimiento de las variables, y ver el contenido de la memoria, los hilos (*threads*) y las variables locales. Como se ve, todo un abanico de posibilidades. Además, las ventanas de depuración permiten el mecanismo "arrastrar y soltar". Por ejemplo, se puede arrastrar el nombre de una variable desde el editor hacia la ventana de seguimiento de variables. A partir de ese momento la variable queda añadida para la depuración.

Optima++ incluye más sistemas para descubrir los errores. El rastreador de memoria permite controlar los desbordamientos de memoria que se producen cuando los objetos se crean y no se eliminan correctamente. Los desbordamientos se manifiestan en cualquier momento de la ejecución del programa, después de haber creado y destruido



La única utilidad externa de Optima++ es este editor de imágenes, ideal para la creación de iconos.

gran cantidad de objetos. Esto hace especialmente complicada la detección del origen del problema. Otro sistema incluido en el entorno de desarrollo es la depuración remota. Es un método para depurar una aplicación en un ordenador mientras se está ejecutando en otro. Ambas máquinas deben estar conectadas en red mediante el protocolo TCP/IP.

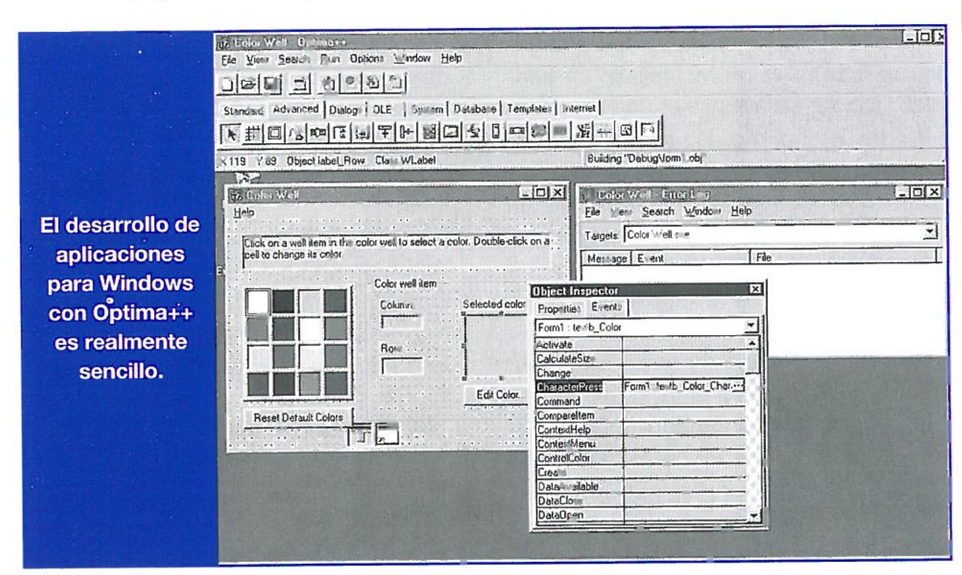

La ventana de código puede mostrar el código completo del formulario, o sólo la parte que interesa en cada momento.

DOCUMENTACIÓN

El paquete incluye un buen número de archivos de ayuda repletos de información. Los archivos más importantes disponibles en la versión comentada son los siguientes:

- *Watcom C Library Reference*: contiene información sobre las funciones de la librería C de Watcom.
- *Watcom C++ Class Library Reference*: similar al anterior. Ofrece toda la información relacionada con las librerías de clases de C++ de Watcom, ordenadas por tipo.
- *DataDirect Database Drivers Help*: ayuda sobre los manejadores ODBC de INTERSOLV. Estos manejadores permiten la utilización de bases de datos mediante SQL.
- *Optima++ Component Library Reference*: información de referencia sobre todos los elementos que forman parte de la librería de componentes de Optima++. Aquí se pueden encontrar todas las propiedades, eventos, tipos de datos y macros disponibles en el paquete.
- *Optima++ Master Help*: es el fichero maestro de ayuda. Reúne la mayor parte de los ficheros de ayuda incluidos en Optima++ junto a otros temas de interés general.
- *Optima++ Programmer's Guide*: se trata de la guía del programador de Optima++. Con ella se pueden aprender los fundamentos de la herramienta, la creación de aplicaciones, los temas avanzados y una breve introducción a C++. Incluye apéndices con la librería de C++, estilo de codificación y bibliografía.
- *Optima++ Samples Guide*: en realidad es una introducción a las posibilidades de Optima++. El producto incluye un alto número de aplicaciones de ejemplo. Son tantas que disponen de su propia guía.
- *Sybase SQL Anywhere Manual*: el manual del producto SQL Anywhere. Incluye la guía del usuario y la guía de referencia de SQL Anywhere.

Además, hay una serie de archivos de ayuda de los controles OCX de Gamesman y la ayuda del editor de imágenes.


El desarrollo de aplicaciones para Windows con Optima++ es realmente sencillo.

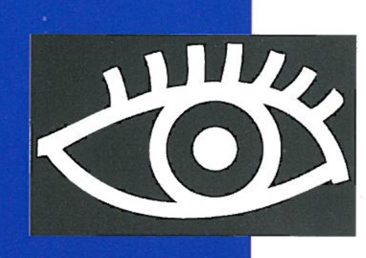
CONSIDERACIONES FINALES

Optima++ se perfila como uno de los mejores entornos para el desarrollo rápido de aplicaciones C++ para Windows. Dispone de un entorno sencillo y poco recargado, pero a la vez muy potente. La posibilidad de incluir código automáticamente con ayuda de la guía de referencia y la técnica "arrastrar y soltar" llevada a su máxima expresión facilitan en gran medida la creación de programas, especialmente para los no iniciados. Incluye un buen soporte para la depuración y el desarrollo de aplicaciones que manejan bases de datos que se ve potenciado con la inclusión del producto Sybase SQL Anywhere, el control DataWindow y los manejadores ODBC de INTERSOLV.

Los creadores de Optima++ han comprendido realmente lo que significa el acrónimo RAD y han sabido reunir lo mejor de otras herramientas en un producto muy interesante.

PROGRAMACIÓN DEL GDI UTILIZANDO LAS MFC 4.0 (II)

Jorge R. Regidor



La independencia de las operaciones gráficas en Windows, debido al GDI, tienen su máximo exponente en el tratamiento de los bitmaps, los cuales internamente tienen toda la información necesaria para ser dibujados de forma independiente del tipo de ordenador y siempre que éste soporte las capacidades gráficas requeridas por la imagen en cuanto a número de colores y resolución.

Dentro de esta segunda entrega de la programación gráfica en Windows con Visual C++ 4.0 se va a hablar acerca de los bitmaps y de las paletas de colores, de tal forma que permitan sentar la base de conocimientos para en el siguiente artículo crear una aplicación que visualice imágenes en formato bitmap (BMP o DIB).

En la primera parte se va a hablar sobre los bitmaps, tanto los independientes de dispositivo como los dependientes, pasando a tratar posteriormente las paletas de colores. A continuación se va a conocer totalmente el formato de los ficheros BMP, lo que permitirá cargar las imágenes desde fichero. Por último, se describirán las estructuras relacionadas con los bitmaps.

QUÉ ES UN BITMAP

La definición suministrada por Microsoft define un bitmap como un objeto gráfico que permite crear, manipular (escalar, rotar, realizar un *scroll* y dibujar) y almacenar imágenes en disco.

Desde el punto de vista de un usuario, un bitmap es un rectángulo en pantalla donde se muestra una imagen, pero desde el punto de vista del programador un bitmap es una serie de estructuras que contienen la información necesaria para mostrar la imagen de una determinada manera dentro de Windows. Con esta información se puede conocer la resolución, el tamaño en pixels, el tamaño de *array* de pixels, la paleta de colores y el *array* de puntos que definen la relación entre cada pixel y las entradas dentro de la paleta de colores.

Existen dos tipos de bitmaps, los dependientes de dispositivo y los independientes de dispositivo. Los DDBs (*Device Dependent Bitmaps*) eran los únicos disponibles en las versiones anteriores a Windows 3.0, pero a partir de esta versión y debido a las limitaciones y dependencia de los dispositivos hardware de los DDBs, surgieron los DIBs (*Device Independent Bitmaps*), que mejoran de forma sobresaliente la fidelidad existente entre las imágenes creadas y mostradas en cada equipo.

Cabe destacar que los bitmaps dependientes de dispositivo están incluidos en las versiones actuales de Windows sólo por compatibilidad, ya que ahora mismo prácticamente sólo se utilizan los bitmaps independientes de dispositivo.

LOS BITMAPS INDEPENDIENTES DE DISPOSITIVO

Los bitmaps independientes de dispositivo contienen la siguientes información sobre los colores y las dimensiones de la imagen:

- La información del formato de color del dispositivo donde se creó la imagen.
- La resolución del dispositivo donde se creó la imagen.
- La paleta de colores que compone la imagen.
- Un *array* de bits que definen cada uno de los pixels que forman la imagen.
- Un indentificador del tipo de compresión utilizada en la codificación del *array* de bits que definen los pixels.

Existen dos variedades de DIBs en función de donde está el origen de la imagen. Por un lado, están los llamados *bottom-up*, DIBs que tienen el origen de la imagen en la esquina inferior izquier-

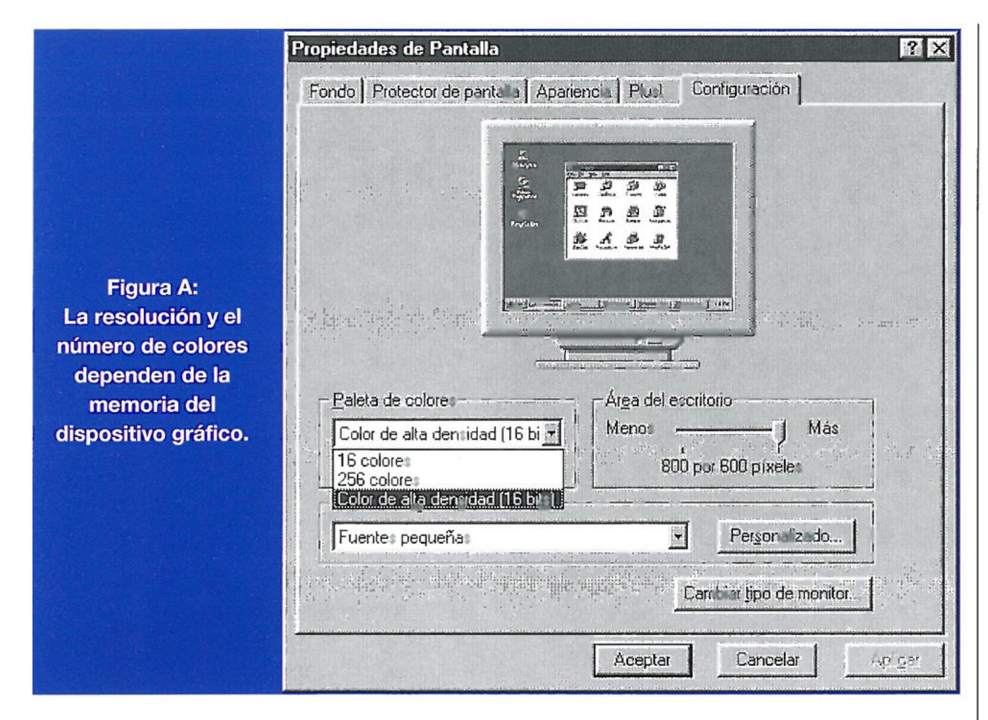


Figura A: La resolución y el número de colores dependen de la memoria del dispositivo gráfico.

da, y los llamados *top-down* que tienen el origen en la esquina superior izquierda. La indicación que define uno u otro de los dos tipos está definida en el campo *biHeight* de la estructura *BITMAPINFOHEADER*. Si el campo comentado es positivo, el tipo de DIB es *bottom-up*, pero si el número es negativo, el tipo de DIB es *top-down*. Como nota especial, hay que indicar que los DIBs *top-down* no pueden ser comprimidos.

El formato del color está definido por los planos de color y los el número de bits por pixel. En Windows, el número de planos de color es siempre 1, pero el número de bits por pixel depende del tipo de adaptador gráfico del sistema. De este modo, un adaptador monocromo tiene un bit por pixel, por lo que cada pixel sólo puede tener dos niveles de color, normalmente blanco y negro. Si el adaptador gráfico es una tarjeta VGA, el número de bits por pixels es 4, lo que permite hasta 16 colores diferentes. Actualmente, todo el mundo conoce los adaptadores gráficos SVGA, que permiten hasta 16 millones de colores, lo que se traduce en 24 bits por pixel. Como el lector habrá podido deducir, el tamaño de la imagen está muy relacionada con el número de bits por pixels, es decir, con el número de colores de la misma. De hecho, esto puede influir en la velocidad del sistema si se selecciona como imagen de escritorio un bitmap grande y con muchos colores, y no se tiene demasiada memoria en el equipo (ver Figura b).

Una aplicación, al cargar una imagen, puede leer el número de bits por pixel del dispositivo al crearse, y además leer el número de bits por pixels soportado por el adaptador gráfico donde se quiere mostrar la imagen y comprobar que dicho adaptador soporta la imagen que se desea cargar. Para leer este parámetro se puede llamar a la función *GetDeviceCaps*, pasándole como segundo argumento la constante *BITSPIXEL*.

La resolución del dispositivo donde se creó la imagen esta contenida en los campos *biXPelsPerMeter* y *biYPelsPerMeter*, que definen la cantidad de pixels por metro en los ejes horizontal y vertical respectivamente. Para calcular los pixels por metro del dispositivo gráfico actual hay que realizar las siguientes operaciones:

1. Llamar a *GetDeviceCaps* pasándole *HORZRES* como segundo parámetro.
2. Llamar a *GetDeviceCaps* pasándole *HORZSIZE* como segundo parámetro.
3. Dividir el retorno de la primera llamada y el retorno de la segunda.

Para el caso vertical hay que realizar las mismas operaciones, pero pasando las constantes *VERTRES* y *VERTSIZE*.

Otra parte importante de los bitmaps independientes de dispositivo es la paleta, que se representa como un *array* de estructuras del tipo RGB, que definen los niveles de color con formato RGB, de todos los colores de los que se compone la imagen. Cada elemento de *array* de la paleta se denomina entrada, y cada pixel de la imagen hace referencia a la entrada de la paleta que define su color.

En Windows 95 y NT, es decir, en los sistemas que utilizan el API Win32, los DIBs de cuatro u ocho bits por pixel (en caso que sean del tipo *bottom-up*) pueden ser comprimidos con el sistema RLE (*Run Length Encoded*). Este sistema utiliza dos bytes para codificar. El primero indica el número de pixels con la misma entrada en la paleta de colores y el segundo define la entrada a la paleta.

Las aplicaciones pueden crear un DIB a partir de un DDB inicializando las estructuras adecuadas y llamando a la función *GetDIBits*. Se puede de nuevo determinar si un dispositivo soporta las llamadas a esta función llamando a *GetDeviceCaps* y pasándole la constante *RI_DI_BITMAP* como bandera de *RASTERCAPS*.

Para mostrar la imagen dentro del dispositivo se debe llamar a la función *SetDIBitsToDevice*, o *StretchDIBits* si se quiere ajustar el tamaño de la imagen al dispositivo. Estas llamadas pueden o deben ser sustituidas por la función *BitBlt* en el caso de que se requieran pintar varias veces la imagen, ya que ésta segunda opción es más rápida.

BITMAPS DEPENDIENTES DE DISPOSITIVO

Como se comentó con anterioridad, los bitmaps dependientes de dispositivo sólo están soportados actualmente por compatibilidad con versiones anteriores, lo que se traduce en que las nuevas aplicaciones no deben de utilizarlos y trabajar con los DIBs.

Los DDBs están íntegramente definidos por la estructura *BITMAP*, la cual define el tamaño de una región rectangular en pixels, el ancho del *array* de entradas a la paleta del dispositivo (lo que lo hace dependiente de las entradas de cada uno de los dispositivos donde se muestren) y el formato del color del dispositivo.

Existen también dos tipos de DDBs, los descargables y los no descargables.

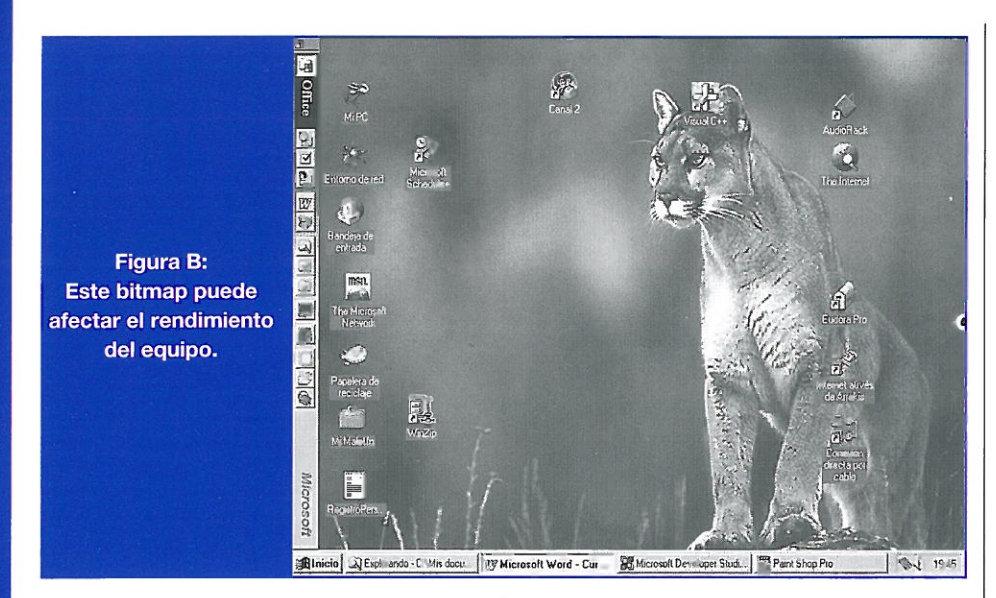


Figura B: Este bitmap puede afectar el rendimiento del equipo.

Los bitmaps descargables pueden ser descargados de la memoria si no están seleccionados en el contexto de dispositivo o si el sistema está bajo de memoria.

LA PALETA DE COLORES

Para definir un color determinado en las tarjetas gráficas hay que indicar los niveles de rojo, verde y azul que contienen. Como en Windows cada nivel de rojo, verde o azul está definido por un BYTE, se pueden conseguir un total de 256*256*256 colores diferentes, aproximadamente 16 millones de colores. Está es la teoría, pero en la práctica la memoria de las tarjetas gráficas está limitada, por lo que normalmente suelen tener un máximo de doscientos cincuenta y seis, treinta y dos mil o sesenta y cuatro mil colores diferentes.

Una paleta de colores es un *array* de valores RGB que definen el conjunto de los diferentes colores que forman parte de una imagen. Dentro de Windows existen dos tipos de paletas, las llamadas paletas lógicas y la paleta del sistema.

La paleta lógica es utilizada por cada aplicación para definir los colores que muestra. La paleta lógica se puede interpretar como la lista de colores necesitados para mostrar una imagen. La paleta del sistema define el conjunto de entradas definidas en un dispositivo, es decir, los colores disponibles en el sistema.

Normalmente, habrá que cargar una imagen, así como las entradas de color necesarias para mostrar dicha imagen. Con estas entradas se deberá crear una paleta lógica y seleccionar ésta dentro de la paleta del sistema para que la imagen se muestre correctamente.

EL FORMATO DE LOS FICHEROS DE BITMAPS

Los bitmaps deben ser guardados con un formato determinado, ya establecido en Windows, y guardar el fichero con la extensión BMP. El formato de estos ficheros consiste en una estructura del tipo *BITMAPFILEHEADER*, seguida de una estructura *BITMAPINFOHEADER*.

La paleta de colores se define como un array de estructuras que contienen los valores RGB que definen cada color

Después de estas dos estructuras hay un *array* de estructuras *RGBQUAD*, que definen las diferentes entradas de colores. Por último, se encuentra el *array* de bits que definen cada uno de los pixels de la imagen.

Los campos de la estructura *BITMAPFILEHEADER* identifican el tamaño del fichero en bytes y el desplazamiento en bytes desde el inicio de la cabecera hasta el primer byte del *array* de bits que definen la imagen.

La segunda estructura *BITMAPINFOHEADER* especifica el tamaño, ancho y alto, de la imagen en pixels, el formato del color (los planos de color y los bits por pixel) del dispositivo donde se creó el bitmap. También se indica si los bits están o no comprimidos, así como el tipo de compresión utilizada. Además se incluye la resolución del dispositivo donde fue creado, el número de bytes del *array* de bits y el número de colores utilizados.

ESTRUCTURAS A UTILIZAR

Para poder emprender con éxito la programación de aplicaciones que manejen DIBs es necesario el conocimiento en detalle de las estructuras que permiten su manejo. El saber qué significa cada uno de los distintos campos permitirá en todo momento saber el porqué de cada uno de los pasos a seguir en nuestras aplicaciones.

LA ESTRUCTURA BITMAPFILEHEADER

El formato de la estructura *BITMAPFILEHEADER* es el siguiente:

```
typedef struct tagBITMAPFILEHEADER
{
 WORD   bfType;
 DWORD  bfSize;
 WORD   bfReserved1;
 WORD   bfReserved2;
 DWORD  bfOffBits;
} BITMAPFILEHEADER;
```

donde:

- *bfType* indica el tipo de fichero. Este campo debe tener el valor BM. De lo contrario se podría asegurar que no es un fichero BMP.
- *bfSize* indica el tamaño en bytes del fichero.
  Los campos *bfReserved1* y *bfReserved2* son campos reservados y ambos deben de tener el valor 0.
- *bfOffBits* indica el número de bits desde el inicio del fichero hasta el origen de los bits que definen la imagen.

Como se puede observar, la información de esta estructura tiene poco que ver con la imagen en sí, pero sí tiene que ver con el formato y el tamaño del fichero.

LA ESTRUCTURA BITMAPINFOHEADER

El formato de la estructura *BITMAPINFOHEADER* es el siguiente:

```
typedef struct tagBITMAPINFOHEADER
{
 DWORD biSize;
 LONG  biWidth;
 LONG  biHeight;
 WORD  biPlanes;
 WORD  biBitCount
 DWORD biCompression;
 DWORD biSizeImage;
 LONG  biXPelsPerMeter;
 LONG  biYPelsPerMeter;
 DWORD biClrUsed;
 DWORD biClrImportant;
} BITMAPINFOHEADER;
```

donde:

- *biSize* es el tamaño de la estructura en bytes.
- *biWitdh* es el ancho de la imagen en pixels.
- *biHeight* es el alto de la imagen en pixels.
- *biPlanes* indica el número de planos de color. Siempre debe ser 1.
- *biBitCount* indica el número de bits por pixel. Este valor puede ser 1, 2, 4, 8, 15, 24 o 32 en función del número de colores de la imagen.
- *biCompression* indica si el *array* de bits está comprimido o no, y en que formato. Los posibles valores son los siguientes:

- ***BI_RGB***: Indica que los bits no están comprimidos.
- ***BI_RLE8***: Indica que los bits están comprimidos con el formato RLE8.
- ***BI_RLE4***: Indica que los bits están comprimidos con el formato RLE4.
- ***BI_BITFIELDS***: Indica que los bits no están comprimidos y que la tabla de colores está definida por tres valores *DWORD* que definen los valores RGB.

- *biSizeImage* es el tamaño en bytes de la imagen. Si el campo *biCompressed* es *BI_RGB*, este valor debe ser 0.
- *BiXPelsPerMeter* indica el número de pixels por metro en el eje X del dispositivo donde se creó la imagen.
- *BiYPelsPerMeter* indica el número de pixels por metro en el eje Y del dispositivo donde se creó la imagen.

BITMAPFILEHEADER
BITMAPINFOHEADER
RGBQUAD
BITS DE LA IMAGEN

Figura B: Este bitmap puede afectar el rendimiento del equipo.

- *biClrUsed* indica el número de colores usados por la imagen.
- *biClrImportant* indica el número de colores importantes de la imagen. Si este campo es cero, indica que todos los colores son importantes.

Como se puede observar, ésta es la estructura más importante del fichero, ya que define todas las propiedades gráficas de la imagen. Es de aquí de donde se debe obtener la información necesaria para observar si el dispositivo gráfico permite o no mostrar la imagen en cuestión.

LA ESTRUCTURA RGBQUAD

El formato de la estructura *RGBQUAD* es el siguiente:

```
typedef struct tagRGBQUAD
{
 BYTE  rgbBlue;
 BYTE  rgbGreen;
 BYTE  rgbRed;
 BYTE  rgbReserved;
} RGBQUAD;
```

donde:

- *rgbBlue* indica el nivel de color azul que contiene esta entrada.
- *rgbRed* indica el nivel de color azul que contiene esta entrada.
- *rgbAzul* indica el nivel de color azul que contiene esta entrada.
- *rgbReserved* es un campo reservado y su valor debe de ser cero.

Esta estructura es cada uno de los elementos del *array* de colores utilizados por la imagen. Como se puede observar, cada nivel de color está definido por un byte, lo que implica que existen 256 tonos de cada color básico.

LA ESTRUCTURA LOGPALETTE

El formato de la estructura *LOGPALETTE* es el siguiente:

```
typedef struct tagLOGPALETTE
{
 WORD   palVersion;
 WORD   palNumEntries;
 PALETTEENTRY palPalEntry[1];
} LOGPALETTE;
```

donde:

- *palVersion* indica la versión de la estructura de la paleta de colores. Actualmente la versión es la 0x300.
- *palNumEntries* indica el número de entradas que forman la paleta lógica.
- *palPalEntry* es un *array* de elementos *PALETTEENTRY* que definen cada una de las entradas.

Como se puede ver, esta estructura es la base de las paletas lógicas, y debe ser inicializada con los valores obtenidos desde los valores *RGBQUAD* del fichero BMP.

Hay que destacar que las entradas de colores deben ponerse de mayor importancia a menor, ya que al colocar los colores en la paleta del sistema los primeros tienen prioridad sobre los últimos.

LA ESTRUCTURA PALETTEENTRY

El formato de la estructura *PALETTEENTRY* es el siguiente:

```
typedef struct tagPALETTEENTRY
{
 BYTE peRed;
 BYTE peGreen;
 BYTE peBlue;
 BYTE peFlags;
} PALETTEENTRY;
```

donde:

- *peRed* es el nivel de rojo en la entrada.
- *peGreen* es el nivel de verde en la entrada.
- *peBlue* es el nivel de azul en la entrada.
- *peFlags* especifica cómo va a ser utilizada la entrada en la paleta. Puede ser uno de los siguientes valores:

- *PC_EXPLICIT*: Especifica que la palabra baja de la entrada es un índice de la paleta del dispositivo hardware. Permite ver el contenido de colores del dispositivo hardware.
- *PC_NOCOLLAPSE*: Indica que esta entrada debe ser insertada en una entrada no utilizada de la paleta del sistema, aunque en ésta paleta exista una entrada idéntica.
- *PC_RESERVED*: Indica que esta entrada va a cambiar sus valores RGB frecuentemente. Sirve para indicar a otras aplicaciones que no fijen una de sus entradas a ésta, ya que va a variar continuamente.
- NULL: Se especifica una entrada normal en la paleta del sistema.

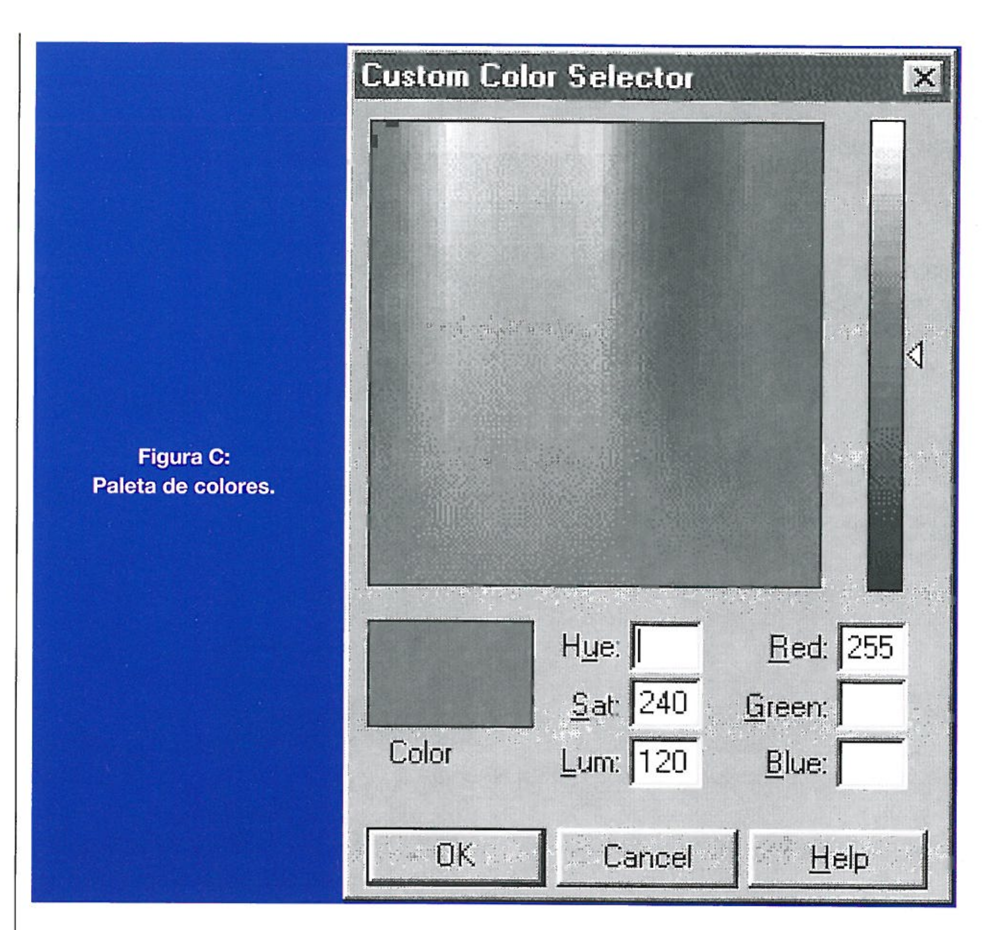


Figura C: Paleta de colores.

PROCESO DE CARGA DE UN DIB

Aunque este proceso se va a ver con todo detalle en el siguiente artículo, se va a realizar una breve descripción del proceso a seguir para cargar un bitmap desde un fichero y mostrarlo en una ventana.

Lo primero, y parece obvio, es abrir el fichero donde se encuentra la imagen. Después se debe leer la estructura *BITMAPFILEHEADER* y comprobar que el campo *bmfHeader* realmente es BM. Después de esto, se debe reservar memoria global para almacenar la estructura *BITMAPINFOHEADER* y las diferentes estructuras *RGBQUAD*, y además para los bits que definen la imagen.

A posteriori, se debe crear una paleta lógica con las mismas entradas como colores use la imagen y copiar los datos de color de las estructuras *RGBQUAD* a las estructuras *PALETTEENTRY*. Normalmente, en el campo *peFlags* se pone *NULL*.

A partir de aquí, uno de los caminos puede ser seleccionar la paleta lógica creada como paleta del sistema y crear un bitmap dependiente del dispositivo a partir del DIB cargado en memoria. Como la paleta de colores del sistema es ahora la que estaba en el fichero, la imagen se mostrará con los colores correctos.

CONCLUSIÓN

En este artículo se ha pretendido mostrar la información necesaria para entender qué son los bitmaps, sus dos principales tipos, así como el formato de los ficheros DIB y las estructuras necesarias para mostrar un DIB en una ventana. Además se ha pretendido introducir conceptos relacionados con las paletas de colores, las cuales son la parte más importante del proceso de carga de DIBs, ya que permiten mapear los colores necesarios para mostrar la imagen.

PRÓXIMA ENTREGA

En el siguiente número se va a realizar una aplicación que visualice bitmaps independientes de dispositivo, lo que permitirá utilizar el formato BMP y sus estructuras asociadas. El rumbo del artículo va a estar íntimamente relacionado con la aplicación, ya que se piensa que esta aplicación debe ser tratada en detalle.

Además, y fruto de la gran cantidad de memoria requerida para cargar un DIB, se van a tratar algunas de las funciones para reservar memoria global en Windows. Este tipo de memoria es ideal para reservar bloques grandes de memoria y que puedan ser descargados a disco, con el objetivo de minimizar la ocupación de la memoria, un recurso normalmente escaso y que se debe tener especial cuidado en su tratamiento.

CONCEPTOS DE PROGRAMACIÓN: PROCEDIMIENTOS

Juan Manuel y Luis Martín

Los lenguajes de programación estructurados facilitan la tarea de programar, ya que permiten dividir ésta en tareas más pequeñas, especializadas y, por tanto, fáciles de acometer. Además, la detección y corrección de errores resulta mucho más cómoda y sencilla. Otra importante ventaja es la posibilidad de realizar procedimientos que pueden ser utilizados por otros programas, sin necesidad de ser creados de nuevo.

Visual Basic dispone de dos tipos principales de procedimientos:

- Procedimientos de Evento: Se ejecutan como respuesta a un evento y están definidos por el propio Visual Basic. Sólo se incluyen en los módulos de formulario, ya que siempre van asociados a formularios y controles.

- Procedimientos Generales: Se invocan desde el código de otros procedimientos, y son creados por el programador. Este tipo de procedimientos se incluyen en los módulos de formulario, módulos estándar y módulos de clase.

En artículos anteriores ya se ha trabajado con procedimientos de evento, y se describió la forma en que pueden escribirse los procedimientos de evento a partir de las plantillas de código proporcionadas por Visual Basic. Los procedimientos generales son similares en su aspecto y modo de ejecución a los procedimientos de evento. La principal diferencia entre ambos radica en que los procedimientos de evento se ejecutan como respuesta a un evento o por invocación directa en el código, mientras que los procedimientos generales deben ser siempre invocados por el programador dentro del código. Por tanto, si en el código no se invoca de forma explícita un procedimiento general, éste nunca llegará a ejecutarse.

Por otro lado, atendiendo a la sintaxis y la funcionalidad de los procedimientos, Visual Basic dispone de tres formatos distintos para su realización, que pueden ser clasificados de la siguiente forma:

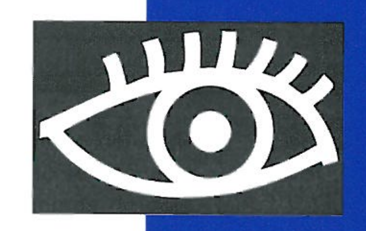
- Procedimientos Sub (o subrutinas): Conjuntos de instrucciones que realizan una determinada tarea.

- Procedimientos Function (o funciones): Conjuntos de instrucciones, igual que las subrutinas, pero que además devuelven un valor a la instrucción que las ha invocado.

- Procedimientos Property: Conjunto de instrucciones que permiten crear y manipular propiedades.

Los procedimientos de evento son siempre subrutinas, ya que no devuelven ningún valor. Por ello, la sintaxis de su cabecera incluye siempre la palabra reservada Sub.

Los procedimientos generales deben ser siempre invocados por el programador



En artículos anteriores se ha podido comprobar que Visual Basic es un lenguaje de programación estructurado, ya que el código se encuentra dividido en fragmentos, denominados procedimientos, que realizan actividades concretas. En esta entrega se analizará el funcionamiento de los distintos tipos de procedimientos soportados por Visual Basic, así como sus diferentes variantes y particularidades.

CREACIÓN Y DECLARACIÓN DE PROCEDIMIENTOS GENERALES

Los procedimientos generales proporcionan al programador una herramienta que le permite realizar tareas específicas. Una vez definido el procedimiento, éste puede ser invocado cuantas veces se desee a lo largo del código para realizar la tarea para la que ha sido diseñado.

Los procedimientos generales se sitúan en la sección General de los módulos de formulario, módulos estándar y módulos de clase. El cuadro 1 muestra los pasos a seguir para crear un procedimiento general. Una vez realizadas estas operaciones, la ventana de código mostrará una plantilla de procedimiento, compuesta por la primera y última líneas del procedimiento, entre las que se deberá introducir el código del mismo, tal y como se muestra en la figura 2.

Los procedimientos en Visual Basic, ya sean de evento o generales, subrutinas o funciones, tienen la misma estructura. Para todos ellos, la primera línea de código es el encabezamiento, y define el tipo de procedimiento y los argumentos que se le deben pasar. Las siguientes líneas constituyen el cuerpo, y estarán compuestas por declaraciones de variables e instrucciones ejecutables. La última línea es una sentencia especial que indica el final del procedimiento.

En el caso de los procedimientos generales de tipo subrutina, la sintaxis puede resumirse de la siguiente forma:

[Private|Public] [Static] Sub NombreSubrutina[(ListaParámetros)]
[Instrucciones]
End Sub

La primera palabra clave indica si el procedimiento va a ser privado (Private) o público (Public). En el primer caso, el procedimiento sólo podrá ser invocado desde el módulo donde ha sido declarado. En el segundo caso, el procedimiento podrá ser invocado desde cualquier módulo de la aplicación. Si se omite esta palabra, los procedimientos se declaran públicos por defecto. Si, además, se incluye la palabra clave Static, todas la variables declaradas dentro del procedimiento serán estáticas.

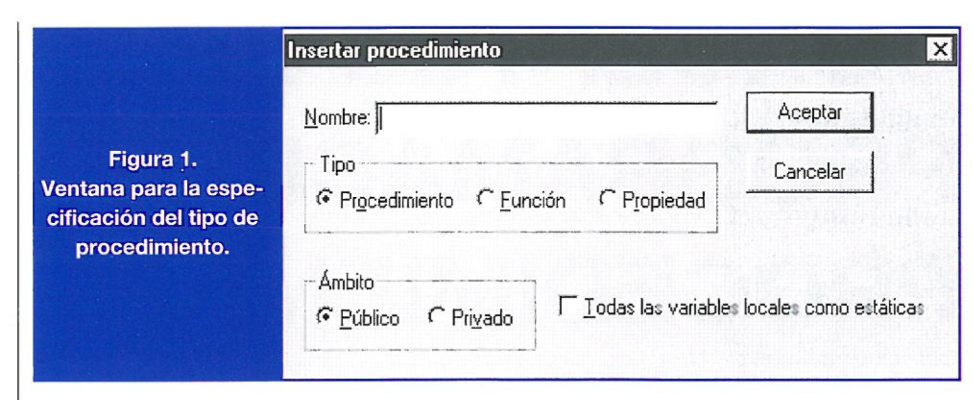


Figura 1. Ventana para la especificación del tipo de procedimiento.

La última parte de la cabecera la constituye la lista de parámetros. Se trata de una lista con los nombres de las variables donde se recibirán los valores pasados como argumentos desde la llamada al procedimiento, según se verá mas adelante.

El cuerpo de los procedimientos está compuesto tanto por declaraciones de variables locales como por instrucciones ejecutables, aunque siempre en ese orden. Las sentencias ejecutables serán las que describan las distintas acciones que realiza el procedimiento. El final de los procedimientos de tipo subrutina se indica mediante la sentencia End Sub.

En el caso de los procedimientos generales de tipo función, la sintaxis es muy similar, presentando sólo pequeñas diferencias:

[Private|Public] [Static] Function NombreFunción[(ListaParámetros)] [As Tipo]
Instrucciones]
End Function

La principal diferencia que presentan las funciones respecto a las subrutinas es que las primeras devuelven un valor a la instrucción que las ha invocado. Por tanto, es posible declarar el tipo de datos que devuelven, utilizando la cláusula As al final de la cabecera.

Las funciones devuelven un valor a la instrucción que las ha invocado

LLAMADAS A SUBRUTINAS Y FUNCIONES

La forma de realizar una llamada a un procedimiento desde el código depende tanto del tipo de procedimiento de que se trate como de su ubicación dentro de la aplicación, es decir, si se encuentra en el mismo módulo desde el que se realiza la llamada o en otro.

En el caso de los procedimientos Sub, la llamada constituye una instrucción independiente por sí misma, no pudiendo ser incluida como parte de una expresión. Esto es debido a que las subrutinas no devuelven ningún valor que pueda ser utilizado para evaluar una expresión. Para llevar a cabo la llamada es posible utilizar dos tipos de sintaxis distintos:

- Llamada explícita: La llamada se realiza mediante la sentencia Call, indi-

CUADRO 1: PASOS A SEGUIR PARA CREAR UN PROCEDIMIENTO GENERAL.

1. Abrir la ventana de código del módulo al que se desea añadir el nuevo procedimiento.
2. Seleccionar la opción *Procedimiento* del menú *Insertar*. De esta forma, se desplegará un cuadro de diálogo como el mostrado en la figura 1.
3. Introducir el nombre del procedimiento en el cuadro *Nombre*.
4. Seleccionar entre los botones *Procedimiento*, *Función* o *Propiedad* en el marco *Tipo*, según se desee crear un procedimiento *Sub*, *Function* o *Property*, respectivamente.
5. Seleccionar entre *Público* y *Privado* en el marco *Ámbito*, dependiendo del ámbito que vaya a tener el procedimiento.
6. Hacer clic sobre el botón *Aceptar*.

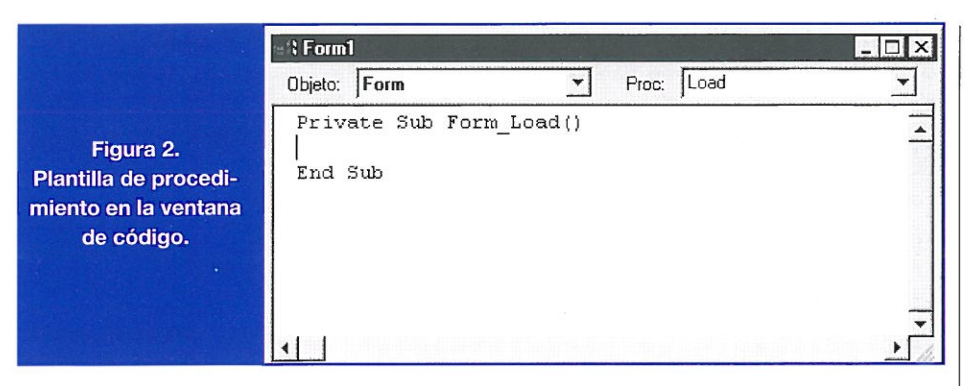


Figura 2. Plantilla de procedimiento en la ventana de código.

cando a continuación el nombre del procedimiento y los argumentos, separados por comas y encerrados entre paréntesis.

- Llamada implícita: la llamada se realiza simplemente indicando el nombre del procedimiento, como si se tratase de una sentencia, seguido de los argumentos, separados por comas y sin paréntesis.

'Declaración de dos subrutinas
Sub MiSubrutina(Cad As String, Num As Integer, Resp As Boolean)
Sub TuSubrutina

'Llamada explícita con argumentos
Call MiSubrutina("Buenos Días", Num1, True)
'Llamada explícita sin argumentos
Call TuSubrutina
'Llamada implícita con argumentos
MiSubrutina "Buenos Días", Num1, True
'Llamada implícita sin argumentos
TuSubrutina

En el caso de las funciones, como devuelven un valor a la instrucción que las invoca, la llamada puede realizarse de forma implícita desde dentro de una expresión que utilice su valor de retorno. Sin embargo, también pueden invocarse de forma explícita, como si se tratase de una instrucción independiente por sí misma. En este último caso, el valor devuelto es descartado.

'Declaración de dos funciones
Function MiFuncion(Cad As String, Num As Integer) As String
Function TuFuncion As Integer

'Llamada implícita con argumentos
Valor = Upper(MiFuncion("Buenos Días", Num1))
'Llamada implícita sin argumentos
Valor = Str(TuFuncion)
'Llamada explícita con argumentos
Call MiFuncion "Buenos Días", Num1
'Llamada explícita sin argumentos
Call TuFuncion

Para llamar a un procedimiento, ya sea una subrutina o una función, es necesario que éste se encuentre en el mismo módulo, salvo que haya sido declarado público. En este último caso la sintaxis de la llamada depende del tipo de módulo donde se ha definido el procedimiento.

- Si se ha definido en un módulo de formulario, es necesario indicar el nombre del formulario delante del nombre del procedimiento, separado de éste por un punto.

- Si se ha definido en un módulo estándar, sólo será necesario indicar el nombre del módulo si existe otro procedimiento declarado con el mismo nombre.

- Si se ha definido en un módulo de clase o en otra instancia de un formulario, es necesario indicar el nombre de la variable que contiene el objeto de dicha clase.

La figura 3 muestra un diagrama explicativo de la sintaxis en cada uno de los tipos de llamadas anteriores.

PASO DE PARÁMETROS

Normalmente, las llamadas a procedimientos llevan lo que se denomina argumentos. Se trata de datos suministrados por la instrucción que realiza la llamada, y que necesitará el procedimiento para poder llevar a cabo su misión. El paso de argumentos al procedimiento permite que en distintas llamadas se trabaje con distintos datos o se realicen distintas tareas.

A la hora de hablar de argumentos, debe hacerse una distinción. Si se habla de ellos desde el punto de vista de la llamada, se denominan argumentos o parámetros reales. Si se habla desde el punto de vista de la recepción, se denominan parámetros o parámetros formales.

Realmente, los parámetros son variables locales del procedimiento que se utilizan dentro del cuerpo de éste para definir los cálculos y acciones que debe realizar el procedimiento. Sin embargo, sólo es necesario declararlos en la cabecera del procedimiento. Así, por ejemplo, dentro de una función que calcula el factorial de un número será necesario utilizar una variable que represente a dicho número, con el fin de llevar a cabo los cálculos necesarios.

Los argumentos son datos suministrados por la instrucción que realiza la llamada

El número y el tipo de los argumentos en la llamada debe ser el mismo que el de parámetros en la declaración. Para ello, en la cabecera del procedimiento deben declararse todos los parámetros que pueden recibir su correspondiente argumento en la llamada. Así, debe especificarse una lista de variables, separadas entre sí por comas, y encerradas todas ellas entre paréntesis. Cada elemento de la lista puede presentar varias cláusulas, según la siguiente sintaxis:

[Optional][ByRef\ByVal][ParamArray] NombreVariable [()][As Tipo]

Tanto el nombre de la variable como su tipo de datos siguen las mismas

Cuadro 2: Ejemplo de función con una matriz como argumento.

```
'Función que suma los elementos de una matriz
unidimensional de enteros
Function Suma(Matriz() As Integer) As Integer
  Dim Indice As Integer, Total As Integer
  Total = 0
  For Indice = LBound(Matriz) To UBound(Matriz)
    Total = Total + Matriz(Indice)
  Next Indice
  Suma = Total
End Function

'Ejemplo de llamada a la función de suma
Dim Numeros(1 To 10) As Integer
...
Print Suma(Numeros()
```

reglas que ya fueron analizadas para cualquier otra variable. Es posible declarar cualquiera de los tipos de datos de Visual Basic, con la única excepción de las cadenas de tamaño fijo.

Sub MiSubrutina(Num As Integer, Cad As String, Dec As Single)
Sub MiSubrutina(Num%, Cad$, Dec!)
...
MiSubrutina 10, "Hola", .345

También es posible pasar matrices como argumentos. Para ello, tanto el argumento en la llamada como el parámetro correspondiente en la declaración deben ir acompañados de dos paréntesis vacíos, (). El cuadro 2 muestra un ejemplo de función para sumar los elementos de una matriz unidimensional de enteros.

La cláusula Optional delante del parámetro permite indicar si el argumento correspondiente es opcional. Si se omite esta cláusula será obligatorio especificar un argumento en la llamada. Además, una vez utilizada esta cláusula para un parámetro, el resto de los parámetros situados a su derecha en la declaración también deben llevarla.

Sub MiSubrutina(Optional Num As Integer, Optional Cad As String, _
Optional Dec As Single)

MiSubrutina
MiSubrutina 10
MiSubrutina 10, "Hola"
MiSubrutina 10, , .345

Para comprobar dentro de un procedimiento si un argumento ha sido o no especificado, puede utilizarse una función especial que devuelve True cuando el argumento ha sido omitido y False en caso contrario.

IsMissing(NombreParámetro)

La cláusula ParamArray indica que el parámetro es una matriz opcional de elementos Variant. Por tanto, dicho parámetro debe ir acompañado de los dos paréntesis. Esta cláusula sólo puede utilizarse con el último parámetro de la declaración, y no puede ir acompañada de las cláusulas Optional, ByRef o ByVal. De esta forma, es posible utilizar un número variable de argumentos en la llamada. Cada uno de estos argumentos quedará situado por orden en los distintos elementos de la matriz. El cuadro 3 muestra un ejemplo de función que suma un número variable de enteros pasados como argumentos.

Finalmente, las palabras clave ByRef y ByVal delante del parámetro permiten especificar si el argumento correspondiente será pasado por referencia o por valor, respectivamente. Por defecto, el paso de parámetros a un procedimiento en Visual Basic se realiza por referencia. Esto quiere decir que, en realidad, lo que se le está pasando al procedimiento no es el valor del argumento, sino la dirección que ocupa en memoria. El procedimiento trabajará directamente sobre esa posición de forma que, si se altera el contenido del parámetro dentro del cuerpo del procedimiento, el valor del argumento también resultará alterado.

La mejor forma de comprender este concepto es mediante un sencillo ejemplo. En éste, la alteración realizada sobre el parámetro dentro del procedimiento se ve reflejada en la variable que actúa como argumento en la llamada.

'Declaración de subrutina que incrementa el argumento
Sub Incrementa(N As Integer)
N = N + 1
End Sub
'Llamada a la subrutina de incremento
Num = 2
Incrementa Num
Print Num 'Resultado: 3

Los argumentos pasados a un procedimiento pueden ser variables, constantes, literales o expresiones. Evidentemente, sólo en el primer caso es posible alterar el valor del parámetro correspondiente dentro del procedimiento. En el resto de los casos, las alteraciones realizadas en las variables involucradas no serán tenidas en cuenta.

Num = 2
*Incrementa 2 * (Num - 1)*
Print Num 'Resultado: 2

ByRef y ByVal especifican si el argumento será pasado por referencia o por valor

Las funciones de Visual Basic sólo permiten devolver un valor. Sin embargo, mediante la utilización del paso de argumentos por referencia, es posible hacer que éstas devuelvan más de un resultado. En este caso, deben pasarse por referencia tantas variables como resultados sean necesarios. El cuerpo del procedimiento debe ocuparse de cargarlas con los valores apropiados. El cuadro 4 muestra un ejemplo de función de este tipo.

Cuando delante de la declaración de un parámetro se incluye la cláusula ByVal, el paso del parámetro se realiza por valor, en vez de por referencia. En este caso, lo que en realidad llega al parámetro del procedimiento es el valor del argumento especificado en la llamada, y no su dirección de memoria. De esta forma, el parámetro se comportará como una variable nueva e independiente, que ha sido inicializada al valor del argumento. En este tipo de llamada es posible alterar el valor del parámetro dentro del cuerpo del procedimiento, sin que por ello se vean reflejados los cambios en el argumento de la llamada.

'Declaración de subrutina que incrementa el parámetro
Sub Incrementa(ByVal N As Integer)
N = N + 1
End Sub

'Llamada a la subrutina de incremento
Num = 2
Incrementa Num
Print Num 'Resultado: 2

Si en la declaración no se ha especificado ninguna de las cláusulas ByRef y ByVal, los argumentos se pasan por referencia. Sin embargo, en este caso, es posible forzar desde la llamada a que el paso se realice por valor. Para ello, será necesario encerrar cada uno de los argumentos de la llamada entre paréntesis.

'Declaración de subrutina que incrementa el argumento
Sub Incrementa(N As Integer)
N = N + 1
End Sub

'Llamada a la subrutina de incremento, forzando el paso por valor
Num = 2
Incrementa (Num)
Print Num 'Resultado: 2

En una llamada a un procedimiento, la asignación de los valores de cada argumento a su correspondiente parámetro se realiza por el orden en que éstos se sitúan. Así, a cada elemento de la lista de argumentos en la llamada le corresponde el elemento de la lista de parámetros del procedimiento que ocupa la misma posición. Este tipo de paso de argumentos se dice que utiliza argumentos posicionales.

Sin embargo, Visual Basic también permite realizar llamadas a procedimientos de forma que los argumentos sean pasados en cualquier orden. Para ello, en la lista de argumentos debe especificarse el nombre del argumento, seguido de los caracteres dos puntos e igual (:=) y la expresión cuyo valor se le desea asignar. Este tipo de paso de argumentos se dice que utiliza argumentos con nombre.

Print Mid(Start := 4, String := "ABCDEF", Length := 2) 'Resultado: DE
Print Mid(String := "ABDCEF", Start := 4) 'Resultado: DEF

No todas las sentencias y funciones de Visual Basic pueden trabajar con argumentos con nombre. Para determinar cuales pueden hacerlo, así como el nombre de los correspondientes argumentos, puede utilizarse la descripción que aparece al seleccionarlas en el Examinador de objetos. Así, por ejemplo, los métodos de la librería Objetos de Visual Basic (VB) no admiten argumentos con nombre, mientras que las funciones de las librerías Visual Basic para Aplicaciones (VBA) y Objetos de Acceso a Datos (DAO) sí los admiten.

Visual Basic también permite la utilización de argumentos con nombre para las subrutinas y funciones definidos por el usuario. En este caso, el nombre de cada argumento debe ser el del parámetro correspondiente en la definición del procedimiento.

Sub Hipotenusa(Cateto1 As Single, Cateto2 As Single) As Single
Print Sqr(Cateto1 ^ 2 + Cateto2 ^ 2)
End Sub
...
Hipotenusa Cateto1 := 10, Cateto2 := 5

La utilización de procedimientos de propiedades permite crear propiedades personalizadas

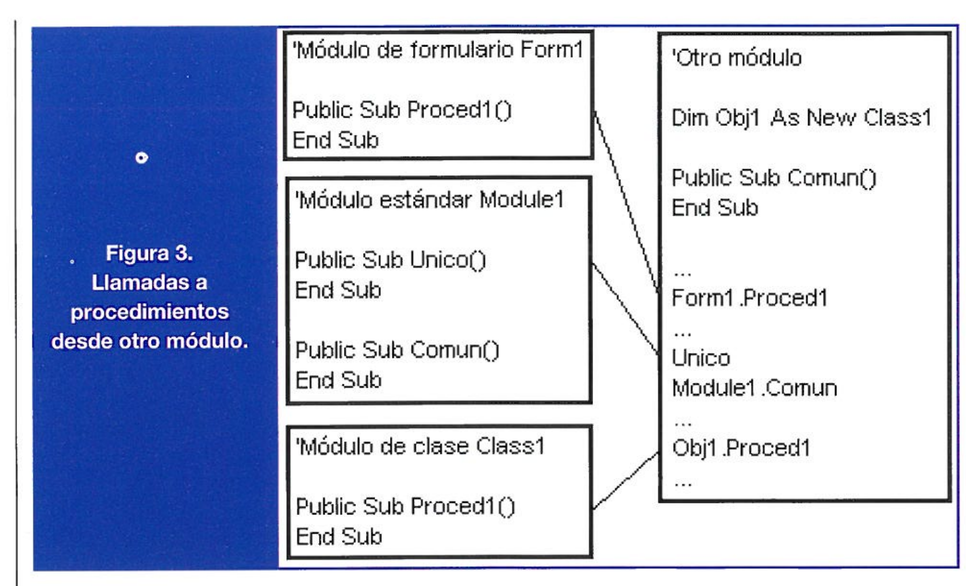


Figura 3. Llamadas a procedimientos desde otro módulo.

CUADRO 3: EJEMPLO DE FUNCIÓN CON UN NÚMERO VARIABLE DE ARGUMENTOS.

```
Función que suma un número variable de enteros
Function Suma(ParamArray Numeros()) As Integer
  Dim Indice As Integer, Total As Integer
  Total = 0
  For Indice = LBound(Numeros) To UBound(Numeros)
    Total = Total + Numeros(Indice)
  Next Indice
  Suma = Total
End Function

'Ejemplo de llamada usando cinco argumentos
Print Suma(1, 2, 3, 4, 5)          'Resultado: 15
```

RETORNO DE SUBRUTINAS Y FUNCIONES

El retorno desde una subrutina o una función viene indicado por las sentencias End Sub y End Function, respectivamente. Cuando se realiza una llamada a un procedimiento, éste toma el control, y sus instrucciones se ejecutan hasta encontrar una de las sentencias anteriores. En ese momento se produce la salida del procedimiento y la devolución del control al punto donde se realizó la llamada. En el caso de las subrutinas, pasará a ejecutarse la siguiente instrucción a la de la llamada.

Sin embargo, el caso de las funciones es algo distinto. Cuando la llamada se realiza dentro de una expresión, y

una vez finalizada la ejecución de la función, ésta es sustituida por el valor devuelto. Seguidamente, se evaluará la expresión, utilizando dicho el valor.

Para especificar el valor que debe ser devuelto por una función, debe utilizarse una instrucción de asignación dentro del cuerpo de la función. De esta forma, se asignará el valor que se desea devolver al nombre de la función, tal y como si se tratase de una variable. Cuando el programa llegue a la última línea, el valor que será devuelto será el último que haya sido asignado.

```
'Función que devuelve un mensaje según un valor lógico
Function SiNo(Resp As Boolean) As String
   If Resp Then
     SiNo = "La respuesta es afirmativa"
   Else
     SiNo = "La respuesta es negativa"
   End If
End Function
```

CUADRO 4: EJEMPLO DE FUNCIÓN QUE DEVUELVE VARIOS VALORES POR REFERENCIA.

```
'Función que calcula las dos soluciones de una ecuación de segundo grado, devolviéndolas en las variables X e Y.

Sub Ecuacion2(ByVal A As Single, ByVal B As Single, _
  ByVal C As Single, ByRef X As Single, ByRef Y As Single)
  'Si existen soluciones reales
  If B ^ 2 - 4 * A * C > 0 Then
    X = (-B + Sqr(B ^ 2 - 4 * A * C)) / (2 * A)
    Y = (-B - Sqr(B ^ 2 - 4 * A * C)) / (2 * A)
  Else
    X = Null
    Y = Null
  End If
End Sub
```

Los procedimientos de propiedades permiten crear propiedades de sólo lectura

Si no se asigna ningún valor de retorno a la función, ésta devolverá un valor por defecto. Este valor es asignado por Visual Basic de forma automática, según el tipo de datos de la función. Así, una función numérica devolverá 0, una función de cadena devolverá una cadena vacía (""), y una función de tipo Variant devolverá un valor Empty.

Visual Basic también proporciona una sentencia especial que permite abandonar la ejecución de un procedimiento anticipadamente. Se trata de las sentencias Exit Sub y Exit Function. Cuando durante la ejecución del procedimiento se llega a una sentencia de este tipo, se produce la salida inmediata del procedimiento, devolviéndose el control a la instrucción que lo invocó, y sin ejecutarse el resto del procedimiento.

```
Sub Mensaje(Optional Texto As String)
   If IsMissing(Texto)
     Exit Function
   Endif
   Print Texto
End Function
```

CREACIÓN DE PROPIEDADES PERSONALIZADAS

Como se ha mencionado anteriormente, tanto la misión como el funcionamiento de los procedimientos de propiedades es algo distinto del de las subrutinas y funciones. Mediante la utilización de este tipo de procedimientos es posible crear propiedades personalizadas en los módulos, dotándolas de un determinado comportamiento.

La creación de propiedades personalizadas puede hacerse simplemente definiendo variables públicas en los módulos, de forma que pueda accederse a éstas desde cualquier otro módulo. Sin embargo, mediante la utilización de procedimientos de propiedades es posible hacer que se ejecute un cierto código cada vez que se establece u obtiene el valor de la propiedad a la que representan. Además, la utilización de procedimientos de propiedades permite crear propiedades de sólo lectura, así como asociarles cadenas de ayuda que pueden ser mostradas mediante el Examinador de objetos.

Para la creación de propiedades personalizadas, Visual Basic dispone de tres tipos distintos de procedimientos de propiedades:

- Procedimientos Property Let: Establecen el valor de la propiedad.
- Procedimientos Property Get: Devuelven el valor de la propiedad.
- Procedimientos Property Set: Establecen la propiedad mediante una referencia a un objeto

En cualquiera de los casos, la sintaxis de los procedimientos de propiedades puede resumirse de la siguiente forma:

```
[Public|Private][Static] Property {Get|Let|Set} NombrePropiedad
    (([ListaParámetros]) [As Tipo]
    [Instrucciones]
  End Property
```

Las cláusulas Public, Private y Static tienen el mismo significado que para el caso de las subrutinas y funciones. Igualmente, es posible especificar una lista de parámetros que representarán a los argumentos pasados al procedimiento en la invocación de éste.

La llamada a los procedimientos de propiedades se realiza cuando se establece o se obtiene el valor de la propiedad a la que representan. Dependiendo del tipo de procedimiento de propiedad (Let, Get o Set), la sintaxis de la llamada será distinta.

Para procedimientos Property Let, la sintaxis es similar a la de asignación de valores a una propiedad:

```
[Objeto.]NombrePropiedad[(ListaArgumentos)] = Expresión
```

En el caso de los procedimientos Property Get, la sintaxis es similar a la de obtención del valor de una propiedad:

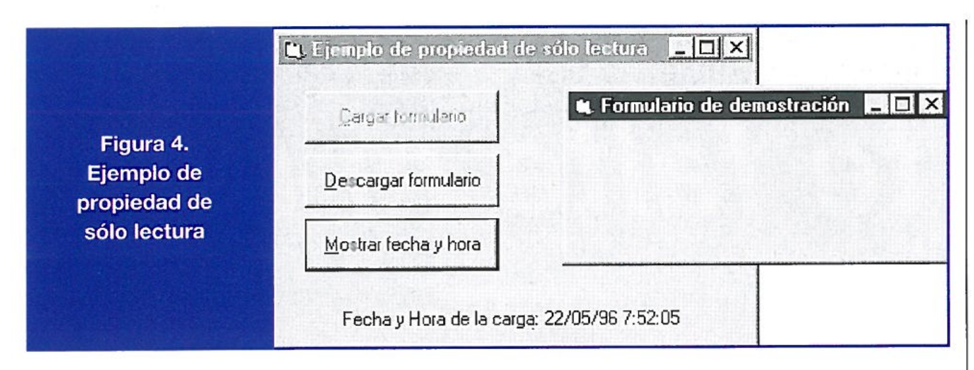


Figura 4. Ejemplo de propiedad de sólo lectura

Variable = [Objeto.]Nombre Propiedad[(ListaArgumentos)]

Finalmente, para los procedimientos Property Set, la sintaxis es similar a la del establecimiento de una referencia a un objeto:

Set [Objeto.]NombrePropiedad[(ListaArgumentos)] = Objeto

Los procedimientos de propiedades suelen trabajar en grupos, permitiendo así ejecutar un cierto código tanto cuando se obtiene el valor de la propiedad como cuando se establece. Para ello, se definen procedimientos Property Get, Property Let y, en su caso, Property Set con el mismo nombre, que será el de la variable a la que representan.

Normalmente suele definirse también una variable a nivel de módulo, que será la encargada de almacenar el valor actual de la propiedad. Este valor será modificado o devuelto por los procedimientos de propiedades una vez que se haya ejecutado el código necesario. Este código suele encargarse de verificar el valor que se desea establecer o de preparar el valor que debe ser devuelto.

La especificación de la lista de argumentos de los procedimientos de propiedades se realiza de forma muy similar a la del resto de procedimientos. Sin embargo, existen ciertas restricciones.

Cuando se declaran procedimientos Property Get, Property Let y Property Set con el mismo nombre, es decir, representando a una misma propiedad, los parámetros definidos en las cabeceras deben tener los mismos nombres y pertenecer al mismo tipo de datos. Sin embargo, los procedimientos Property Let y Property Set deben tener un parámetro más que el procedimiento Property Get.

Property Get NombrePropiedad(Param1, Param2, ... , ParamN) As Tipo
Property Let NombrePropiedad(Param1, Param2, ... , ParamN, ParamN+1)
Property Set NombrePropiedad(Param1, Param2, ... , ParamN, ParamN+1)

Este último parámetro de más, representado por ParamN+1 en la sintaxis anterior, será en el que se reciba el valor que se desea asignar a la propiedad. De esta forma, en la llamada al procedimiento, el argumento correspondiente a este parámetro figurará en el lado derecho de la sentencia de asignación.

Sólo debe definirse un procedimiento de propiedad, que será siempre de tipo Property Get

El último parámetro de los procedimientos Property Let y Property Set debe pertenecer siempre al mismo tipo de datos devuelto por el procedimiento Property Get. Además, cuando se haya definido el procedimiento Property Set, el tipo de datos de este argumento debe ser siempre un objeto o un Variant. Esto se debe a que su valor siempre será asignado mediante la sentencia de asignación Set, que trabaja con referencias a objetos.

```
Property Get Prop(A As Integer, B As Boolean) As String
End Property

Property Let Prop(A As Integer, B As Boolean, C As String)
End Property

Prop(10, True) = "Hola"
Print Prop(2, False)
```

Aunque es posible utilizar múltiples argumentos en los procedimientos de propiedades, en la práctica su utilización sólo se justifica para la creación de matrices de propiedades. En ese caso, además del parámetro que representa el valor de la propiedad, deben especificarse los índices del elemento a manejar dentro de la matriz.

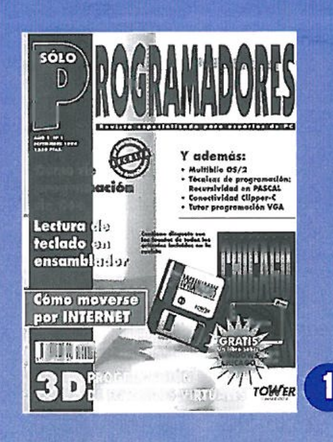
Para crear matrices de propiedades, en la sección de declaraciones del módulo debe declararse una matriz. Si el número de elementos de la matriz es variable, el procedimiento de propiedad debe encargarse de redimensionarla para evitar que el índice quede fuera de rango.

El siguiente fragmento de código muestra cómo puede crearse y gestionarse una matriz unidimensional de propiedades de tipo String mediante procedimientos Property.

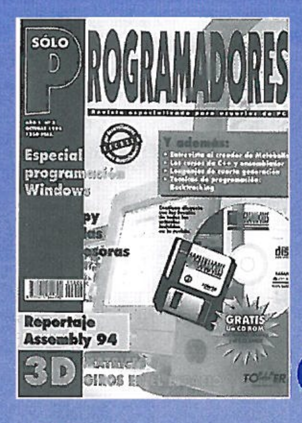
```
Dim Matriz() As String, Indice As Integer
Property Let Demo(Ind As Integer, Cad As String)
  If Ind > UBound(Matriz) Then
    ReDim Preserve Matriz(Ind)
  End If
  Matriz(Ind) = Cad
End Property
Property Get Demo(Ind As Integer) As String
  If Ind >= LBound(Matriz) And Ind á= UBound(;atriz) Then
    Demo = Matriz(Ind)
  Else
    Demo = "ERROR"
  End If
End Property
```

Los procedimientos de propiedades también permiten crear propiedades en los módulos cuyo valor no puede establecerse, aunque sí obtenerse. A este tipo de propiedades se les denomina propiedades de sólo lectura. Para crear propiedades de sólo lectura debe decla-

¡Que no te falte ni uno!



1



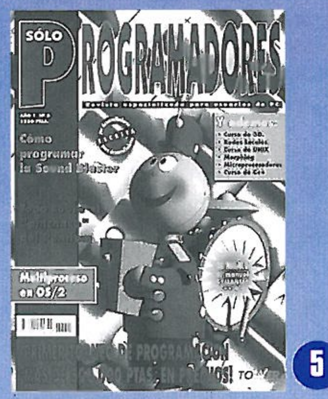
2



3



4



5



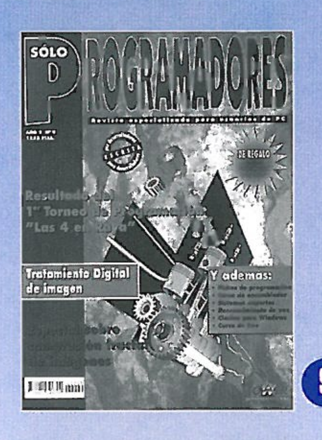
6



7



8



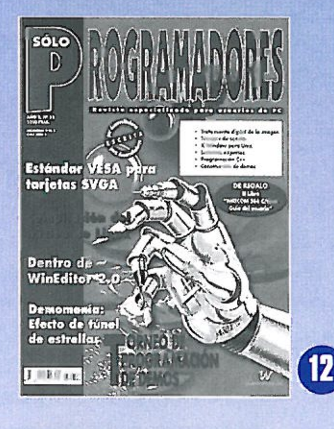
9



10



11



12



13



14



15



16

rarse una variable privada a nivel de módulo, que contendrá el valor de la propiedad. De esta forma, sólo será posible acceder a dicha variable desde el módulo donde ha sido declarada. Además, sólo debe definirse un procedimiento de propiedad, que será siempre de tipo Property Get. Este procedimiento será el encargado de devolver el valor cuando sea consultado desde el código de otro módulo.

El cuadro 5 muestra los pasos de un ejemplo con una propiedad de sólo lectura para un formulario. Se trata de la propiedad Cargado, que devuelve la fecha y hora en que fue cargado el formulario. La fecha y hora de la carga del formulario de demostración se asigna a la variable FechaHora en el procedimiento de evento Load. Al tratarse de una variable privada, no será posible acceder a ella desde el primer formulario salvo a través del procedimiento Property Get, que devuelve su valor.

ARRANQUE Y TERMINACIÓN DE UNA APLICACIÓN

Cuando se arranca una aplicación, uno de sus formularios es siempre cargado y visualizado. A este formulario se le denomina formulario inicial. Cuando se carga este formulario, el primer procedimiento de evento que se ejecuta es el correspondiente al evento Initialize. Igualmente, cuando se descarga dicho formulario, el último procedimiento de evento que se ejecuta es el correspondiente al evento Terminate.

Cuando la aplicación consta de varios formularios, el formulario inicial es el procedente de Form1. Sin embargo, es posible cambiar el formulario inicial por cualquier otro del proyecto. Para ello, debe seleccionarse Opciones en el menú Herramientas, y acceder a la ficha Proyecto. Dentro de ésta, la lista desplegable Formulario inicial permite seleccionar uno de los formularios del proyecto.

Visual Basic también permite que las aplicaciones arranquen sin que se cargue ningún formulario inicial. En este caso, el código que se ejecutará inicialmente será el de un procedimiento general de un módulo estándar, que siempre debe llamarse Main. Además, este procedimiento debe ser de tipo subrutina.

Para que el arranque se produzca con el procedimiento Sub Main, debe seleccionarse la opción Sub Main en la lista desplegable Formulario inicial de la ficha Proyecto anteriormente mencionada.

Para descargar un formulario debe utilizarse la sentencia Unload. Si esta sentencia se encuentra en el código del propio formulario, puede añadirse la palabra reservada Me para indicar que el formulario a descargar es el actual (Unload Me).

Visual Basic también dispone de una sentencia que permite finalizar la ejecución de una aplicación de forma inmediata. Se trata de la sentencia End que, aunque descarga de la memoria todos los formularios de la aplicación, no permite ejecutar ningún tipo de código más, incluidos los de los procedimientos de evento Unload, Terminate y QueryUnload.

Es posible arrancar una aplicación desde un procedimiento general llamado Main

Para que una aplicación finalice completamente, es necesario que todos sus formularios se encuentren descargados y no se ejecute ningún código. Sin embargo, en determinadas ocasiones puede parecer que la aplicación ha finalizado sin haberlo hecho realmente. Esto puede ser debido a la existencia de formularios cargados en la memoria, pero ocultos. Para evitarlo, es conveniente descargar todos los formularios de la aplicación.

Cuadro 5: Ejemplo de propiedad de sólo lectura.

1. Añadir tres botones de comando, una etiqueta y un segundo formulario vacío, situándolos de forma similar a como se muestra en la figura 4.

2. Especificar en tiempo de diseño las siguientes propiedades para ambos formularios:

Form1	Name	frmSoloLec
	Caption	Ejemplo de propiedad de sólo lectura
Command1	Name	cmdCargar
	Caption	&Cargara formulario
Command2	Name	cmdDescar
	Caption	&Descargar formulario
	Enabled	False
Command3	Name	cmdMostrar
	Caption	&Mostrar fecha y hora
	Enabled	False
Label1	Name	lblMensaje
	Option	Formulario de demostración descargado
	Alignment	2 - Center
Form2	Name	frmDemo
	Caption	Formulario de demostración

3. Escribir los siguientes procedimientos de evento para el primer formulario:

```
Private Sub cmdCargar_Click()
  frmDemo.Show
  lblMensaje.Caption = "Formulario de demostración
cargado"
  cmdCargar.Enabled = False
  cmdDescar.Enabled = True
  cmdMostrar.Enabled = True
End Sub
Private Sub cmdDescar_Click()
  Unload frmDemo
  lblMensaje.Caption = "Formulario de demostración
descargado"
  cmdCargar.Enabled = True
  cmdDescar.Enabled = False
  cmdMostrar.Enabled = False
End Sub

Private Sub cmdMostrar_Click()
  lblMensaje.Caption = "Fecha y hora de la carga: " &
frmDemo.Cargado
End Sub
```

4. Escribir las siguientes declaraciones y procedimientos para el segundo formulario:

```
Private FechaHora As Date

Private Sub Form_Load()
  FechaHora = Now
End Sub

Public Property Get Caragado() As Date
  Cargado = FechaHora
End Property
```

5. Salvar el primer formulario en el archivo

SOLOLEC1.FRM, el segundo en SOLOLEC2.FRM y el proyecto en SOLOLEC.VBP.

INTRODUCCIÓN A LOS COMPONENTES TAPI

Carlos Arias Alonso

La evolución de las telecumunicaciones y su aplicación a la informática han conducido hoy en día a una integración entre estas dos materias. Como resultado, las casas de software fabricantes de sistemas operativoshan creado soluciones para proveer herramientas de desarrollo de software para realizar aplicaciones que integren servicios telefónicos.

Dado que es materialmente imposible condensar en un solo artículo todos los conocimientos relativos a esta interface de programación, se comenzará dando una visión general, imprescindible para entender la arquitectura TAPI, dejando para otros artículos el análisis en detalle de los distintos elementos.

QUÉ ES TAPI

Microsoft, a partir de Windows 95, ha integrado como parte del sistema operativo servicios de acceso telefónico a redes mediante TAPI.

TAPI (Telephony application programming interface) es una API que propoporciona, mediante una librería de acceso dinámico (DLL), soporte para la programación de aplicaciones telefónicas. TAPI está soportado por Windows 3.X, Windows 95 y Windows NT 4.0. La versión actual es la 1.4 para Windows 95 y la 2.0 para Windows NT 4.0.

La idea básica que subyace en esta interfaz de programación es sencilla: limitarse a establecer la comunicación, dejando la transferencia de datos a las funciones estándar del SDK de Windows.

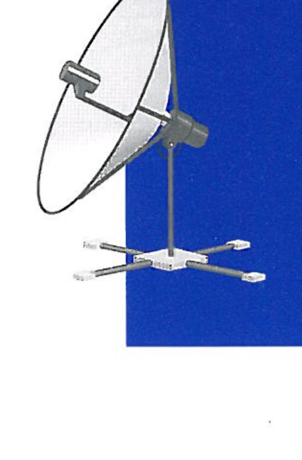
Entre los ejemplos de utilidades que utilizan TAPI y que vienen junto con Windows 95 figuran los siguientes: el driver de fax y de Compuserve que se cargan junto con Microsoft Exchange, el marcador telefónico Phone Dialer y el Hyperterminal para acceso a BBS.

Lo primero en lo que uno piensa cuando se habla de dar soporte al acceso telefónico a redes es en la transmisión de datos y voz mediante modems de 28.800 o 33.600 bps conectados a una línea POST convencional, realizando llamadas al exterior y aceptando llamadas entrantes. Este es un concepto muy pobre. TAPI va mucho más allá, permitiendo la conexión a gran velocidad mediante tarjetas RDSI (Red digital de servicios integrados), control remoto de ordenadores, transferencia de llamadas con dispositivos que soporten esta capacidad, administración de conferencias, autentificación de la llamada de origen (caller ID), etc. Mediante tarjetas RDSI del tipo BRI (interface de ratio básico) se pueden conseguir velocidades de transferencia de 128 Kbps y mediante tarjetas tipo PRI (interface de ratio primario) se puede superar ampliamente esta velocidad.

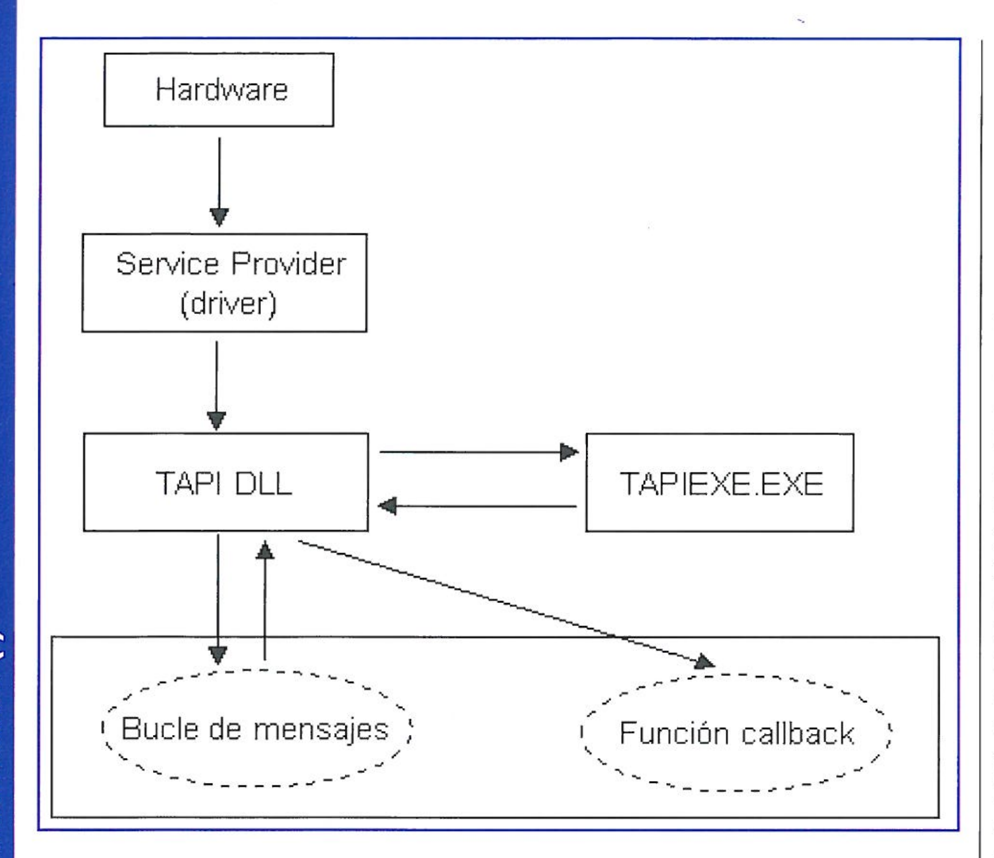
TIPOS DE DISPOSITIVO

Según las capacidades que tengan los distintos periféricos telefónicos que se pueden conectar a un ordenador, TAPI los diferencia en dos clases:

- Dispositivos de línea: es una representación abstracta de un aparato físico que conecta el ordenador a una línea telefónica. El ordenador tiene control parcial del dispositivo. Por ejemplo, un teléfono sencillo auxiliado por un módem que actúa como marcador, TAPI lo ve como un dispositivo de línea.



Durante esta serie de artículos se explicará con detalle todos los secretos que conciernen a la interfaz de programación telefónica de Microsoft, acompañando a los artículos con una serie de ejemplos para clarificar los conceptos que se vayan exponiendo.



- Dispositivos telefónicos: es la representación abstracta de un dispositivo con todas las capacidades de un teléfono, teniendo control total sobre el mismo. Este teléfono puede ser perfectamente un programa emulador de teléfono que utilice el módem, micrófono y altavoz del ordenador.

LOS SERVICIOS TAPI

TAPI pone a disposición del programador tres tipos de servicios: básico, suplementario, y extendido.

- Servicios básicos: Son normalmente soportados por todos los dispositivos telefónicos. Proporciona la capacidad para poder marcar un número de teléfono o descolgar ante la recepción de una llamada. Normalmente, todos los recursos de línea soportan este tipo de servicio.

- Servicios suplementarios: Están disponibles para dispositivos especiales que soporten la administración de conferencias, transferencia de llamadas, etc. Son soportados por los dispositivos telefónicos.

- Servicios extendidos: Orientados a los drivers del hardware.

FORMATO DE NÚMERO TELEFÓNICO

En las líneas actuales, tanto en Europa como en Estados Unidos, el número de teléfono que hay que marcar para llamar a un determinado lugar es relativo al sitio de origen de la llamada. Por ejemplo, el número de teléfono 555 6666 de Barcelona variará dependiendo desde donde se realice la llamada. Si la llamda es urbana, se marcará 555 6666. En cambio, si se llama desde Madrid, habrá que marcar 93 555 6666. Para solucionar este problema, TAPI utiliza dos tipos de formatos de números telefónicos para sus operaciones.

- Canonical address format: Describe el número de teléfono con independencia del origen de la llamada. Va precedido del signo "+". Para ello se divide el número destino en varios campos, separando los dígitos que representan el código de pais, código de área y número local.

- Dialable address: Se diferencia del anterior enque no lleva el símbolo "+". Se suele utilizar para marcar una dirección dada sin modificador de ningún tipo. Para la conversión de formatos se puede utilizar la función TranslateAddress.

ARQUITECTURA DE SOFTWARE

Los servicios TAPI son proporcionados a través de una librería dinámica con extensión DLL y de un fichero ejecutable con extensión EXE. La DLL junto con el ejecutable actúan como una capa intermedia entre las aplicaciones y los proveedores de servicio. Se llama proveedor de servicio al driver que actúa de enlace entre el hardware y la DLL de TAPI. Un ejemplo es el UNIMODEM (Universal Modem) suministrado con Windows 95, poniendo a disposición del sistema servicios TAPI para módems compatibles con el estándar HAYES.

Al igual que con los distintos protocolos de red, con TAPI se pueden realizar operaciones síncronas y asíncronas. En las operaciones asíncronas, el fichero ejecutable *TAPIEXE.EXE* se encarga de enviar a la aplicación los mensajes de notificación a través de funciones callback. La programación asíncrona es un tanto compleja, por lo que hay que tener especial cuidado a la hora de su implementación.

TAPI clasifica los dispositivos en dos tipos, de línea y de teléfono

- **Operaciones Síncronas.**

Son la mayoría de las funciones. La llamada a la función retorna cuando se completa la operación, indicando en su valor de retorno si ha tenido éxito o no.

- **Operaciones Asíncronas.**

La llamada a la función retorna sin haber completado la operación. El valor de retorno indica si se inicializó con éxito la operación. Ésta es completada en otra tarea, devolviendo el resultado final mediante una función callback. Estas funciones callback hay que regis-

trarlas cuando se inicializa la librería TAPI.

- **Funciones callback.**

Hay que realizar su implementación con cuidado, dado el mecanismo que utiliza TAPI para notificar sucesos. Su utilización es algo compleja, ya que van auxiliadas por el bucle de mensajes de la aplicación o tarea. A continuación se describen los pasos que sigue Windows para notificar un suceso TAPI a una aplicación:

1º. El driver del proveedor de servicio llama a la DLL cuando desea colocar una notificación.

2º. La DLL envía un mensaje de notificación al ejecutable *TAPIEXE .EXE.*

3º. El ejecutable *TAPIEXE.EXE* llama de nuevo a la DLL,pero esta vez en su propio contexto de ejecución. Esta llamada da instrucciones a la DLL para que envíe un mensaje a la aplicación que está utilizando el dispositivo.

4º. La aplicación procesa el mensaje en su bucle de mensajes, y lo envía de nuevo a la DLL, esta vez en el contexto de la aplicación.

5º. La DLL, ya en el contexto de la aplicación, llama a las funciones callback registradas de la aplicación para notificarle el suceso.

Esta técnica implica que se preste especial atención a la hora de diseñar el bucle de mensajes de la aplicación, ya que la función callback sólo es llamada cuando se procesa el mensaje enviado por TAPI en el bucle de mensajes de la aplicación.

En las aplicaciones multihebra que utilicen llamadas asíncronas hay que tener en cuenta que cada tarea o hebra tiene que tener su propio bucle de mensajes. Esto es así dado que las funciones callback son llamadas en el contexto de la hebra donde se realizó la llamada.

FORMATO DE NÚMERO TELEFÓNICO

Canonical address

+ código_de_pais espacio [(código_de_área) espacio] número_local |subdirección_RDSI ^ nombre_RDSI CRLF

Ejemplo:
+034 (91) 5556666

El número_local puede contener cualquiera de los caracteres de control utilizados en la dialable addres:

Dialable address

Número_de_marcado | subdirección_RDSI ^ Nombre_RDSI CRLF

Ejemplo: 5556666

El número_de_marcado puede llevar los siguientes caracteres modificadores:

! Produce una señal hookflash, colgando por un periodo de tiempo de tiempo de 75 mm antes de continuar marcando

Pp Indica marcación por pulsos.

Tt Indica marcación por tonos.

, Realiza una pausa en la marcación.

Ww Indica al modem que espere al tono de marcado.

@ Indica que espere varios segundos de silendio antes de que continue el marcado.

? Indica que el usuario será interrogado antes de continuar con la marcación

; Debe de ser situado al final de la cadena e indica que la información de marcado se completará más adelante

| Es opcional e indica que los caracteres que haya hasta un + o ^ se refieren a una subdirección de RDSI.

^ Es opcional e indica que los caracteres que haya hasta un CRLF se refieren a un nombre de RDSI.

Función de conversión de número de teléfono:

```
lineTranslateAddress(HLINEAPP hLineApp, DWORD dwDeviceID,
   DWORD dwAPIVersion, LPCSTR lpszAddressIn, DWORD dwCard,
   DWORD dwTranslateOptions, LPLINETRANSLATEOUTPUT lpTranslateOutput)
```

CRLF es la representación del código ASCII Hex (0D) seguido por el ASCII Hex (0A)

CAMPOS DE DATOS

TAPI utiliza como parámetros en sus funciones estructuras de datos de longitud variable. Esto es debido a los campos opcionales que puede recibir una función. Para que TAPI pueda saber el tamaño de la estructura que se le está pasando, se inicializa uno de los campos de la estructura antes de realizar la llamada. Si no se sabe a priori el espacio requerido, se pueden realizar dos llamadas a la misma función. En la primera llamada se pasa el tamaño mínimo de la estructura de datos, devolviendo la función en uno de los campos el tamaño necesario. En la segunda llamada se relocaliza la memoria extra indicada en la llamada anterior. Cuando es la función TAPI la que devuelve un puntero de estructura de datos a la aplicación, se indica la memoria que ha utilizado mediante un tercer campo. El esquema de este tipo de estructuras de datos se muestra en la figura adjunta.

En la documentación de Microsoft son referenciadas estas estructuras como de tipo *"flattened"*. En ellas no se utilizan punteros para apuntar a los parámetros opcionales. El campo de longitud variable es identificado en la estructura por unos parámetros de desplazamiento y de longitud. El desplaza-

Los servicios TAPI se dividen en tres categorías: servicios básicos, servicios suplementarios y servicios extendidos

ESTRUCTURA DE DATOS DE LONGITUD VARIABLE

dwTotalSize	Tamaño total
dwNeededSize	Tamaño necesario
dwUsedSize	Tamaño utilizado
Otros miembros de la estructura	
dwVarItemSize	
dwVarItemOffset	
Otros miembros de la estructura	
dwVarItemSize	
dwVarItemOffset	
Campos de longitud variable	

miento especifica en bytes la posición de comienzo del campo relativa al comienzo de la estructura, la longitud especifica el tamaño del campo en bytes.

Un ejemplo en C++ del mecanismo a seguir para utilizar una variable de este tipo es el siguiente

```
VARSTRUCT * pVarStruct= (VARSTRUCT * ) malloc ( sizeof(pVarStruct) );
pVarStruct->dwTotalSize= sizeof(VARSTRUCT);
tapiStrangeFunc (pVarStruct);
VARSTRUCT * pTemp= (VARSTRUCT * ) realloc (pVarStruct,
 pVarStruct->dwNeededSize);
if (pTemp)
{
pVarStruct= pTemp;
tapiStrangeFunc (pVarStruct);
}
```

PARÁMETROS DE VERSIÓN

Hay varias funciones que necesitan que se les pase como parámetro la versión API con la que se está trabajando. Para obtener el número de versión existen las funciones **lineNegotiateAPI Version** y **phoneNegotiateAPIVersion**. Dependiendo de la función que solicite este parámetro, hay que obtener la versión de una manera u otra. Algunas funciones no necesitan que se obtenga la versión por este sistema, valiendo con pasarles el número más alto de la API con la que se está trabajando.

TAPI utiliza dos tipos de nomenclatura para los números de teléfonos: la canonical address y la dialable address

EL MODELO DE PROGRAMACIÓN

Se explican a continuación los pasos a seguir para establecer una sesión telefónica mediante TAPI.

En La mayoría de las ocasiones el primer paso a seguir será interrogar al usuario sobre los parámetros de conexión tales como el número de teléfono y el tipo de llamada a realizar. Para esta tarea TAPI no dispone de ningún tipo de ayuda de cara al programador, desarrollando el interface de usuario mediante el API estándar de Windows. Por ejemplo, se puede tener utilizar una agenda electrónica en la que el usuario seleccionará el número de teléfono al que se llamará.

En el siguiente paso se le pasa a TAPI mediante un puntero la función *callback* que se utilizará para la notificación de sucesos en las llamadas asíncronas. Esto se hace mediante la función *lineInitialize*, la cual devuelve el número de dispositivos de línea disponibles para la aplicación.

Como tercer paso hay que negociar con TAPI el número de versión a utilizar, tal y como se ha visto anteriormente, y las capacidades del dispositivo llamando a la función *lineGetDevCaps*. Esta última función no es necesaria invocarla para cada nueva llamada, ya que las capacidades del dispositivo serán normalmente las mismas a lo largo de la sesión telefónica.

Después de conocer las capacidades del dispositivo y del número de versión a utilizar, se puede abrir el dispositivo de línea llamando a la función *lineOpen*. Esta función retorna el handle que será utilizado para subsecuentes operaciones. Se pueden indicar distintos privilegios de acceso a la línea mediante uno de los parámetros de la estructura que se pasa a dicha función.

Una vez se tiene el dispositivo de línea abierto la siguiente operación es el establecimiento de la conexión invocando a la función de llamada telefónica *lineMakeCall* o aceptando llamadas entrantes. A partir de este momento se pueden enviar y recibir datos mediante funciones estándar de Windows, no utilizando el juego de instrucciones de TAPI para este cometido.

Por último, cuando se desea terminar la conexión hay que colgar la línea invocando a la función *lineDrop*, cerrar el dispositivo mediante *lineClose* y terminar la sesión TAPI invocando a *lineShutdown*.

CONTENIDO DEL CD-ROM

DEMOS EUSKAL UTILIDADES SHAREWARE, GUÍA-SP

El presente número de Sólo Programadores viene acompañado de un CD-ROM dividido en varios apartados: Algunas de las mejores demos que se presentaron en el último certamen de la Euskal Party, los habituales fuentes y utilidades de los artículos correspondientes a este mes de Diciembre, la primera entrega de la Guía-SP de informáticos, la cual crecerá a medida que los lectores programadores, o sea todos, vayan enviando sus currículos a nuestra redacción, una colección de herramientas y utilidades shareware que más abajo se describen, y los habituales fuentes correspondientes a los artículos de este número 28.

GUÍA-SP DE INFORMÁTICOS

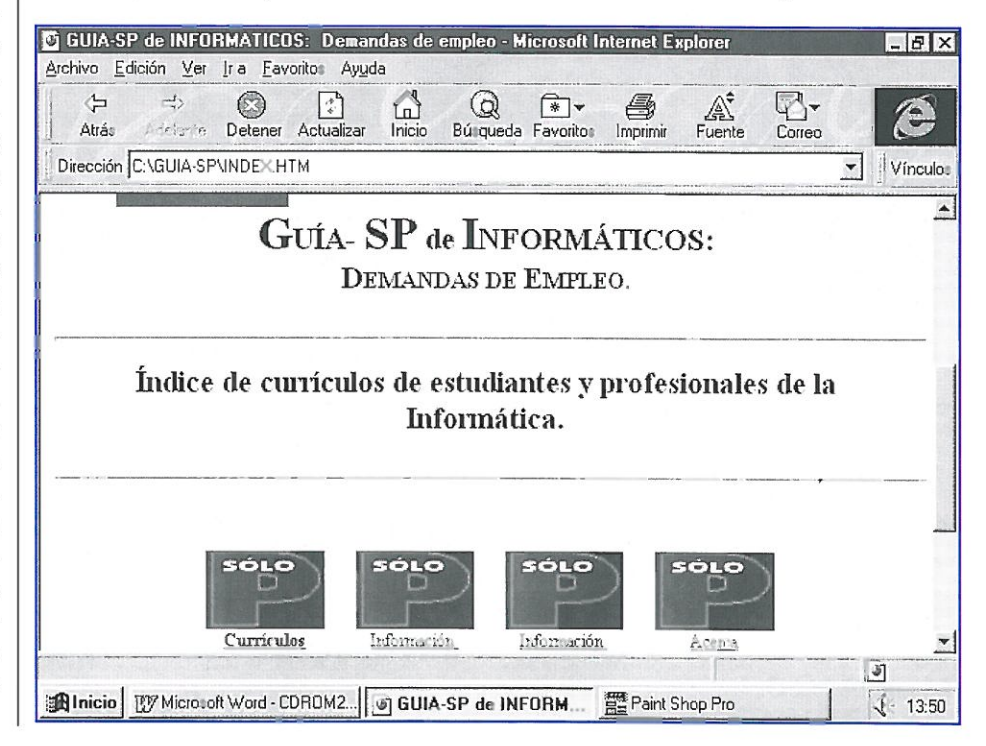
Lo que era un proyecto, ya es realidad. A partir de ahora, aquellas empresas que necesiten programadores tienen una opción más para hallarlos, con la ventaja de que las personas que aparezcan en nuestro CD, irán avaladas por Sólo Programadores. Todos los meses en nuestro CD aparecerá esta guía, periódicamente se irá actualizando la información, el objetivo principal es ayudar a los jóvenes programadores a encontrar el primer empleo. Las empresas, a su vez, tendrán una fuente de currículos disponibles, con muy buenos programadores en potencia. Nada de ofimáticos, de cualquier manera la mayoría de la gente anunciada en la guía tendrá conocimientos de ofimática, pero lo que nosotros queremos fomentar son puestos de trabajo como programadores, ya sean de alto, bajo nivel o multimedia.

La Guía-Sp de informáticos se encuentra en el directorio GUIA-SP, el formato, páginas HTML, es decir se ha diseñado para que el acceso sea a través de navegación off line. Para su vusualización se ha de utilizar un browser como el Internet explorer que se encuentra en el directorio BROWSERS, Netscape o cualquier otro, la primera página a cargar es *index.htm*. Esperamos que aceptéis con agrado esta idea.

EUSKAL PARTY 96

En el directorio EUSKAL se encuentra una colección de demos participantes en la pasada edición de esta party, sabemos la gran aceptación que tienen entre nuestros lectores las demos, de ahí que nos hayamos decidido a darlas a pesar de algunos problemas que presentan algunas en su ejecución. A continuación damos unos consejos que evitarán "reseteos del equipo" :

- Copiar las demos a disco duro, no ejecutar desde el cd-rom.
- Asegurarse de conocer los parámetros correctos de la tarjeta de sonido (Port, IRQ, DMA).
- Conseguir la máxima memoria convencional posible, algunas de las producciones requieren bastante cantidad de memoria base.
- Previa a la ejecución, leer los diferentes ficheros de ayuda e informa-



ción que incorporan la mayoría de las demos.

Las distintas producciones has sido agrupadas en dos subdirectorios correspondientes a las categorías de Demos, Intros 64k e Intros 4k. Son los subdirectorios /DEMOS, y /INTROS4K. Dentro de estos, cada subdirectorio pertenece a una demo en concreto. Esperamos que disfrutéis con estas pequeñas maravillas. Se recuerda que en el número 26, se publicó un reportaje sobre este evento donde aparecía la tabla de ganadores.

UTILIDADES SHAREWARE

En el directorio UTILS se encuentra una colección de utilidades de programación y tutoriales. A continuación se realiza una descripción de las más significativas :

En el subdirectorio AXSDK se encuentra el Microsoft ActiveX development kit, o lo que es lo mismo el kit de desarrollo para ActiveX. Los controles ActiveX son una solución para la creación de objetos que puedan ser insertados en páginas Web principalmente, pero también en aplicaciones Visual Basic y otras que soporten tecnología OLE.

Atención, para la instalación de ActiveX SDK es necesario tener previamente instalado lo siguiente : Internet Explorer 3.1, la última versión del Win32 SDK y Visual C++. Así mismo, requiere Windows 95 o la versión final de NT 4.0

Aquellos que no posean alguno de estos requisitos, podrá instalar alguno de los componentes con el fin de poder visualizar la gran cantidad de información que se proporciona en formato htm, que merece la pena.

Para la instalación :

Crear directorio en disco duro, por ejemplo temp

Copiar el autodescomprimible, activex, a este directorio auxiliar,

descomprimirlo tecleando *activex.exe -d*

Desde Windows ejecutar *setupsdk*.

El programa comenzará el proceso de instalación.

De cualquier manera, se recomienda encarecidamente la lectura de los distintos ficheros con extensión txt que se encuentran en dicho subdirectorio.

En el subdirectorio C (de UTILS) se han agrupado varias utilidades C, destacar las siguientes :

ISAMMGR es una librería para la gestión de bases de datos relacionales para programación C++. Trabaja bajo Dos, Windows y OS/2. Soporta bases de datos relacionales, rápido acceso, acceso simultáneo multi-usuarios con posibilidad de lock a usuarios, hasta 2 millones de registros para ficheros de datos, etc. Para la instalación teclear *Install,* es necesario añadir en el path c :\isammgr\bin si se ha utilizado este directorio (el de instalación por defecto). Una vez completada la instalación con éxito el lector comprobará la existencia de subdirectorios con ejemplos y tutoriales.

TUTORC es un muy completo tutorial de programación en C, se ejecuta mediante el fichero *tutor.exe*.

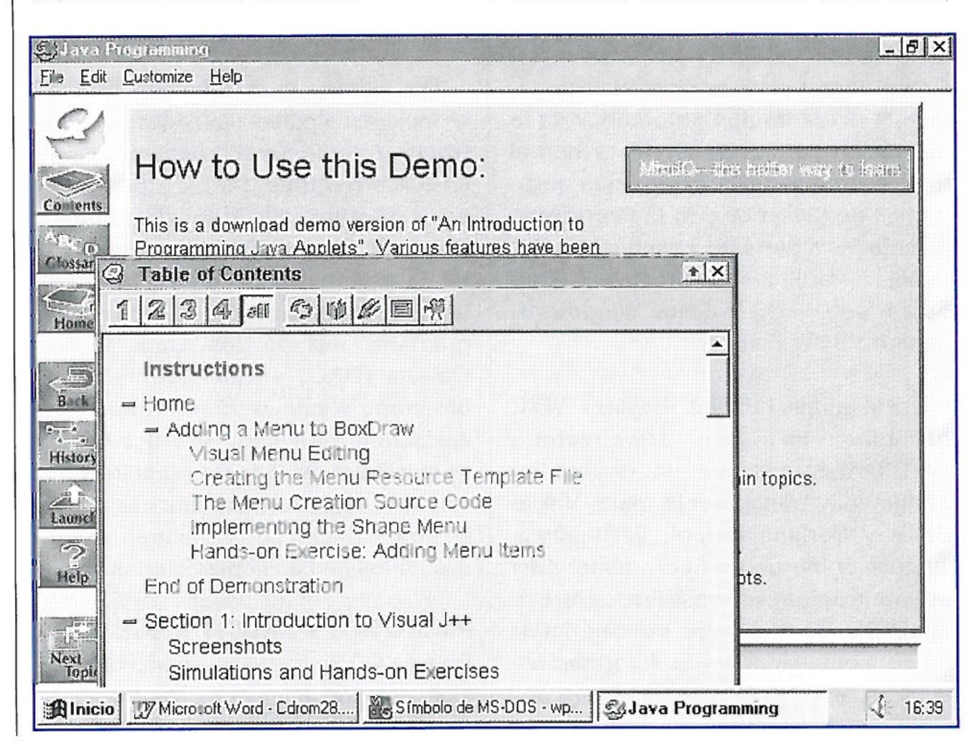
MICRO-C es un pequeño compilador de C, incluye ejemplos y documentación.

En el subdirectorio \JAVATUT de UTILS se encuentran 4 tutoriales relacionados con programación de applets Java de la empresa MindQ. Ideal para neófitos en la materia. Funciona en Windows 95 y Windows 3.X. Para su ejecución lo siguiente, en cada subdirectorio \TUTOX (tuto1, tuto2, tuto3, tuto4) ejecutar *setup*. Dicho programa creará un grupo propio, desde el cual lanzar la aplicación.

En el subdirectorio \PASCAL, una de las estrellas de esta recopilación shareware, Pascal Lite, un completo compilador Pascal de 32 bits para 386/486 y Pentiums de Intel, genera código en modo protegido usando el extensor DOS32. Ideal para practicar con los ejemplos y utilidades que aparecen mensualmente en la sección Cómo programar una demo. Para su ejecución teclear *Instalar*, después será necesario modificar el path en nuestro *autoexec.bat* añadiendo el directorio C:\TMTPL\BIN.

En el subdirectorio \VBASIC se encuentran dos utilidades de gran ayuda para este entorno.

La primera de ellas, VB HelpWritter, situada en el subdirectorio \HELPWRIT,



consiste en un editor de ayudas para Windows que automatiza el proceso de creación de este tipo de ficheros. Como características principales destaca la actualización automática de la aplicación para crear el enlace con el fichero de ayuda, la generación automática de glosarios y la fácil creación de enlaces hipertexto y menús desplegables. Para instalar esta utilidad habrá que ejecutar desde Windows el fichero VBHW.EXE.

La segunda utilidad, Regions VBX, ubicada en el subdirectorio \REGIONS2, consiste en un control multimedia transparente para Visual Basic y Borland Delphi, destinado a situarse sobre un gráfico o texto y así definir regiones sensitivas (ofrece alrededor de 500) que pueden servir como contenedores de información, enlaces, etc... Para instalar este control se debe ejecutar el fichero SETUP.EXE ubicado en dicho sub directorio.

Por último, en el directorio \DELPHI se incluyen algunas utilidades para este lenguaje como Direct Access, situada en el subdirectorio \DIRECT. Se trata de un excelente sustituto del Borland Database Engine (BDE) que proporciona un potente acceso a bases de datos tanto de forma transparente como programada usando los Data Access Objects (DAO) en plataformas de 32 bits como Windows 95 y NT. Se instala ejecutando el fichero SETUP.EXE que se encuentra en el mismo subdirectorio. Además de ésta, otras muchas utilidades para Delphi se encuentran repartidas en los demás subdirectorios.

MICROSOFT FRONT PAGE 97

En este CD se incluye también una versión beta de Microsoft FrontPage. Se trata de una herramienta rápida y sencilla de utilizar que trabaja como cualquier otra aplicación de la familia Office 97 y permite crear y mantener Webs sin necesidad de programar. Como características principales destaca la posibilidad de importar páginas Web y su contenido hacia FrontPage, soporte para lenguaje VBScript y JavaScript y soporte de módulos plug-in de Netscape. La instalación se realiza ejecutando el fichero SETUP.EXE, del directorio FPAGE, desde Windows 95 ó NT y requiere un PC con procesador 486 ó superior, 8 Mb de memoria RAM para Windows 95 y 16 para NT y 30 Mb de disco duro.

ARTÍCULOS

Como viene siendo habitual, el CD-ROM contiene los archivos correspondientes a los artículos de la revista. Se pueden encontrar en los distintos subdirectorios situados dentro del directorio \DISK28.

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

ATHENA WIDGETS

P *Ante todo felicitar por la excelente visión de actualidad de su revista capaz de tocar todos los puntos de interés. Soy Ingeniero de Telecomunicación al que le resta el proyecto fin de carrera. En el mismo desarrollo una herramienta gráfica en X-Window para una gestión de trabajo SNMP (TCP/IP). La programación X-Window es lo suficientemente complicada como para tener que usar los Athena Widgets, sin embargo la documentación sobre estos es muy escasa, por no decir nula. Me gustaría saber dónde puedo conseguir tal información lo antes posible.*

Manuel Pedraza Bueno
(Málaga)

R Es cierto que la documentación existente sobre los Athena Widgets es bastante escasa. Puede que le interese más utilizar Motif, ampliamente documentado y con abundante información disponible en Internet. Pero como su pregunta versa sobre los Athena Widgets, trataremos de ayudarle.

En Internet puede encontrar una pequeña documentación sobre dichos widgets. Pruebe el siguiente URL:

http://www.cs.curtin.edu.au/documentation/X11R5/Xaw/html/index1.html

Si la información contenida en dicho URL no le satisface, o prefiere disponer de un libro sobre el tema, puede que le sirva la siguiente obra:

"X Toolkit: Intrinsics & Athena Widgets R3"
Autor: McCormack y otros
ISBN: 092930604X
Editorial: Silicon Press

Posiblemente tenga dificultades para encontrar este libro en España. Si tiene problemas, intente pedirlo a alguna de las grandes firmas dedicadas a libros de informática en Estados Unidos.

CURSO DE DEMOS

P *Hola, estoy leyendo vuestro curso de demos entrega XV "El rellenado sólido" y lo encuentro muy interesante. De hecho, he estado buscando algún curso de demos que explique en detalle tanto la parte teórica como la práctica. Desafortunadamente esta es la primera y única entrega que tengo sobre este curso. Os escribo para pediros una lista de todos los números atrasados de Sólo Programadores que tengan una entrega de este curso de demos, desde la primera hasta la XV. De este modo podré ir pidiéndolas poco a poco.*

Fernando Carbajo Cruces
(Sevilla)

R No hay problema, amigo Fernando. Aquí tienes una lista de los números que te faltan, junto con el título del tema tratado en el correspondiente artículo de nuestro curso de demos:

Nº 11: "El fuego".
Nº 12: "El túnel de estrellas".
Nº 13: " El plasma".
Nº 14: "Assembly 95: La gran party del PC".
Nº 15: "Los textos".
Nº 16: "Una intro".
Nº 17: "La lente".
Nº 18: "Los sprites".
Nº 19: "El efecto blur motion".
Nº 20: "Introducción a las 3D".
Nº 21: "Shading bobs".
Nº 22: "Ocultación de caras en 3D".
Nº 23: "Las barras de copper".
Nº 24: "Algoritmos fractales".

Puedes pedirlos directamente a la propia editorial por teléfono, fax o bien por correo. Los datos aparecen siempre en las primeras páginas de la revista.

LENGUAJE DE PROGRAMACIÓN

P *Hola, amigos de Sólo Programadores. Acudo a vosotros para ver si podéis asesorarme. Llevo tiempo pensando en informatizar el pequeño negocio familiar que ocupa la mayor parte de mi tiempo. El programa tendría que manejar bases de datos y realizar gran cantidad de cálculos relativamente sencillos. Sólo tengo unas nociones básicas de programación y no sé cuál sería el lenguaje de programación más indicado.*

Ricardo Mata Lozano
(Burgos)

R En este caso lo ideal sería una herramienta de desarrollo rápido de aplicaciones para Windows. Son muy fáciles de usar, y su manejo se aprende con rapidez. Puesto que sus conocimientos de programación son básicos pueden servirle perfectamente Visual Basic o Borland Delphi. Ambos proporcionan unas versiones mejoradas de Basic y Pascal (respectivamente), dos lenguajes fáciles de aprender y que le permitirán hacer lo que necesita.

ACCESO A C DESDE CLIPPER

P *Hola, soy un programador de una empresa de informática. Hace tiempo que estoy intentando conectar el lenguaje C de Borland, concretamente el 2.0, con el Clipper 5.2 de Computer Associates. Mis preguntas son sencillas:*
- ¿Se pueden conectar estos dos lenguajes?
- En caso afirmativo, ¿Qué opciones de compilación tengo que utilizar en el Borland C?
- ¿Qué opciones de compilación tengo que utilizar con Clipper?

Juan A. Vargas
(Elche)

R En principio no hay problema en conectar el lenguaje C con Clipper utilizando los compiladores que usted menciona. Para ello existen tres posibilidades: llamar a un programa externo escrito en C pasando los parámetros por la línea de comandos, utilizar la sentencia CALL de Clipper y utilizando el sistema extendido. La más interesante y también la más complicada es esta última.

El sistema extendido permite realizar llamadas a funciones escritas en C desde un programa Clipper. La llamada desde el programa Clipper mantiene la sintaxis habitual, pero la función en C debe utilizar una serie de funciones especiales para poder comunicar ambos lenguajes. Todas estas funciones están definidas en el archivo EXTEND.API, situado en el directorio INCLUDE de Clipper.

Para recoger los parámetros pasados a la función se utilizan:

int _parinfo (int): nº de parámetros pasados.
int _parinfa (int, unsigned int): nº de elementos de un parámetro de tipo array.
char *_parc (int, ...): parámetro de tipo carácter.
unsigned int _parclen (int, ...): longitud del parámetro de tipo carácter.
int _parni (int, ...): parámetro de tipo entero.
long _parnl (int, ...): parámetro de tipo entero largo.
XDOUBLE _parnd (int, ...): parámetro de tipo numérico doble.
int _parl (int, ...): parámetro de tipo lógico.
char *_pards (int, ...): parámetro de tipo fecha.

Si se quieren cambiar los valores de los parámetros, se utilizarán las siguientes funciones, complementarias de las anteriores:

```
int _storc (char far *, int, ...);
int _storclen (char far *, int, ...);
int _storni (int, int, ...);
int _stornl (long, int, ...);
int _stornd (XDOUBLE, int, ...);
int _storl (int, int, ...);
int _stords (char far *, int, ...);
```

Y por último, para devolver valores se utilizan:

void _retc (char far *): devolución de un carácter.
void _retclen (char far *, unsigned int): devolución de una cadena.
void _retni (int): devolución de un entero.
void _retnl (long): devolución de un entero largo.
void _retnd (XDOUBLE): devolución de un número doble.
void _retl (int): devolución de un valor lógico.
void _retds (char far *): devolución de una fecha.
void _ret (void): no devolver nada.

Un pequeño ejemplo de utilización puede ser una función que calcula el producto de dos números enteros y además intercambia el valor de ambos parámetros:

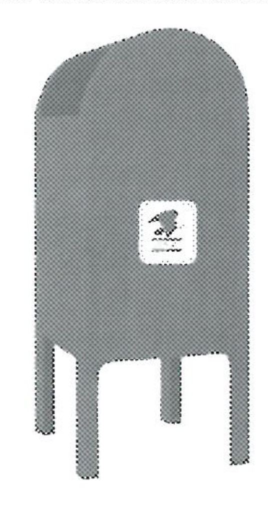
```
// PRUEBA.PRG
procedure main ()
 local a := 3
 local b := 4
 local c := 0
 ? "La variable a vale ", a
 ? "La variable b vale ", b
 c := opera (@a, @b)
 ? "Tras la operación..."
 ? "La variable a vale ", a
 ? "La variable b vale ", b
 ? "El resultado es ", c
quit

// PRUEBAC.C
#include "extend.h"
#include <stdio.h>
CLIPPER opera (void) {
 int aux = _parni(1);
 _storni (_parni(2), 1);
 _storni (aux, 2);
 _retni (_parni(1) * _parni(2));
}
```

Para crear el ejecutable se utilizarían los siguientes comandos:

```
clipper prueba /n /a /w
bcc -c -ml -f- -N- -G -O -Z -Ic:\clipper5\include pruebac.c
rtlink FI prueba, pruebac LIB clipper, cl /NOE
```

En el ejemplo se ha supuesto que el compilador Clipper 5.2 está instalado en el camino C:\CLIPPER5. No olvide actualizar la variable de entorno LIB para que RTLINK sea capaz de encontrar la librería CL.LIB en el directorio LIB donde está instalado Borland C++.



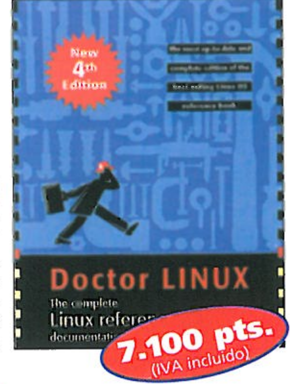
CD-ROMs *muy especiales*

Linux Red Hat Commercial v4.0

¡2 CD-ROMs + libro 230 pgs.!

- El Linux más solicitado en la actualidad.
- Versión oficial 4.0 (i386) de Red Hat Software, Inc.
- Incluye licencia para el servidor X de Metro.
- De fácil instalación y mantenimiento.
- Actualizaciones automáticas vía Internet y CD.

6.810 pts.

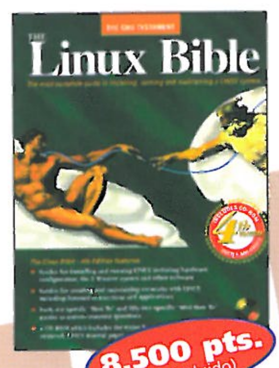


Doctor Linux 4ª edición

¡Libro de 1.600 páginas!

- El complemento ideal para el Linux Red Hat.
- Guías y libros de instalación, configuración, mantenimiento y administración.
- De fácil consulta, cuidada selección y organización.

7.100 pts. (IVA incluido)



Linux Bible 4ª edición

¡CD-ROM + libro 1.800+ pgs.!

- La recopilación de documentación para Linux más completa existente.
- CD-ROM incluyendo el libro completo para búsquedas y consultas.
- Guías y libros de instalación, configuración, mantenimiento y administración.

8.500 pts. (IVA incluido)

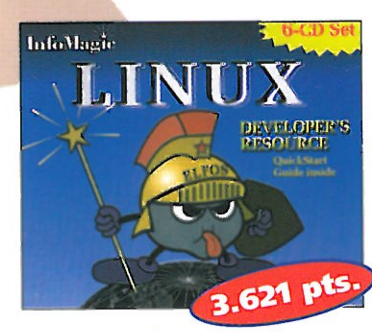
Linux Slackware 96

¡Edición de agosto/96!
¡Ahora kernel 2.0! ¡4 CD-ROMs!



3.621 pts.

- Guía impresa de instalación rápida, 36 páginas.
- Recopilación oficial por Patrick Volkerding.



3.621 pts.

Linux Developer's Resource

¡Edición de septiembre/96!
¡Ahora kernel 2.0! ¡6 CD-ROMs!

¡¡NOVEDAD!! para Linux

Cyclom-8Zo

8 puertos serie • Procesador RISC de 32-bits
Bus PCI • 460 Kbps full-duplex por canal

$449

(precio de promoción para distribuidores y proveedores de Internet)



CD0923
Herramientas para programadores Delphi.
2.155 pts.



CD0929
Ejemplos, código y documentación para Delphi.
2.413 pts.



CD0958
Herramientas para administradores Novell.
3.017 pts.



CD0892
Herramientas de programación Perl.
3.621 pts.



CD0930
Herramientas de programación Visual Basic.
2.413 pts.

Sistemas Operativos/Programación: ***Disponemos del más amplio y especializado catálogo de CD-ROMs de utilidad para programadores y profesionales. Consúlte nuestro Web.***



abc analog, s.l.

abc analog, s.l.
☎ (91) 634 20 00
(91) 634 32 13
FAX (91) 634 47 86
Internet:
http://www.abcnet.es





